Synthesizing Real-time Human Animation by Learning and Simulation

Xizhi Li
The CKC honors school of Zhejiang University
Computer Science Department
email: LiXizhi@zju.edu.cn

Abstract:
Real-time human animation covers a wide range of applications such as virtual reality, video games, web avatars, etc. This paper presents a research framework aimed at synthesizing real-time humanoid animation by integrating the variables of the environment into the controllers of the human body and using a learning and simulation algorithm to calculate and memorize its motion. Both the environment and the human body can be partially or fully controlled by an external user; whereas the motion for the uncontrolled portion will be generated from an internal algorithm. To produce realistic animation, the environment and the body movements are first fully controlled until the animation system has discovered the patterns for the various combinations of the different parts of the body and the environment variables; then only the environment and selected parts of the human body are controlled, the system will generate the motion for the rest. The advantages of the framework are (1) the motion is fairly realistic since it is based on examples. (2) different parts of the human body may act less dependently; e.g. the top of the body might react to other environmental changes other than synchronizing with the bottom of the body. The animation system is included in a computer game engine we developed. Finally, how the research fits into the large context of automatic motion synthesis in the game engine will be discussed.

Key words: Human Animation, Game Engine

The author's note to himself:

2005-1-11: In the last month, I have tried many ways to express ST theory in a formal way, but not very successful. The current article is a tentative description of the ST theory in the form of a motion generation system. Father has backed up this theory with his insightful thesis written over 24 years ago. My manuscripts are in two volumes of scratch papers.

I have compressed f(t) using monotonic functions of its segmentations. The fact that a monotonic function always has its inverse function is very important in the ST theory (even in Father's original manuscripts). The distribution of attentions is still the hardest part in the mathematical formulation. HMM is a model that I have compared with ST1.

I have wanted to first develop a drawing application before implementing the motion system in my game engine. The drawing application will draw ahead of the user in a different color (old lines fade away). After some training the drawing application will be able to draw as the user does.

The title of this article is also motivated in the computer graphic class at ZJU. Sister Ying Liu's thesis [2] also gives me the initiative to carry on the job (character animation engine). The game DOOM3 and Prince of Persia 2 were released shortly before I started writing the article. I compared with them a lot. The thesis of Anim'99[1] provides me with some useful guidance on the subject. Only section 2.3 is important in this paper. All other sections are either not finished or reorganized from my previous work [3, 4] or other articles [1].

Obsolete formulae:

$$f_i\ (T_c+\triangle T) = f_i\ (T+\triangle T), \text{ where } att_i(T)\bullet f_i\ (T)\bullet\triangle T \geq \max\ \{\int_x^{x+\triangle T} att_i(x)\bullet f_i(x)dx \mid 0 \leq x \leq (T_c-\triangle T)\}$$

Contents:

1 Introduction

Real-time human animation covers a wide range of applications such as virtual reality, video games, web avatars, etc. When animating a character, there are three kinds of animation which are usually dealt with separately in a motion synthesis system: (1) local animation, which deals with the motion of its major skeleton (including its global speed), (2) global animation, which deals with the position and orientation of the character in the scene, (3) add-on animation, which includes facial animation and physically simulated animation of the hair, cloth, smoke, etc. This paper mainly concerns about the local animation. Local animation is usually affected by the status of the character (such as a goal in its mind) and its perceptible vicinity (such as terrain and sounds).

The motion of a specified human character can be formulated by a set of independent functions of time, i.e. $\{f_n(t) \mid n = 1,2,...,N; t \in [T_0, +\infty]\}$. These functions or variables typically control over 15 movable body parts arranged hierarchically, which together form a parameter space of possible *configurations* or poses. For a typical animated human character, the dimension of the configuration is round 50, excluding degrees of freedom in the face and fingers. Given one such configuration at a specified time and the static attributes of the human figure, it is possible to render it at real-time with the support of current graphic hardware. Hence, the problem of human animation is reduced to, given $\{f_n(t) \mid n = 1,2,...,N; t \in [T_0, T_c]\}$ and the environment W (or workspace in robotics), computing $\{f_n(t) \mid n = 1,2,...,N; t \in [T_c, T_c + \Delta T]\}$ which should map to the desired and realistic human animation.

The first option to motion generation is to simply play back previously stored motion clips (or short sequences of $\{f_n(t)\}$). The clips may be key-framed or motion captured, which are used later to animate a character. Real-time animation is constructed by blending the end of one motion clip to the start of the next one. To add flexibility, joint trajectories are interpolated or extrapolated in the time domain. In practice, however, they are applied to limited situations involving minor changes to the original clips. Significant changes typically lead to unrealistic or invalid motions. Alternatively, flexibility can be gained by adopting kinematic models that use exact, analytic equations to quickly generate motions in a parameterized fashion. Forward and inverse kinematic models have been designed for synthesizing walking motions for human figures [5, 6]. There are also several sophisticated combined methods [1, 7] to animate human figures, which generate natural and wise motions (also motion path planning) in a complex environment. Motions generated from these methods exhibit varying degrees of realism and flexibility.

This paper presents a different motion generation framework aimed at synthesizing real-time humanoid animation by integrating the variables of the environment into the controllers of the human body and using a learning and simulation algorithm to calculate and memorize its motion.

Both the environment and the human body can be partially or fully controlled by an external user; whereas the motion for the uncontrolled portion will be generated from an internal algorithm. To produce realistic animation, the environment and the body movements are first fully controlled until the animation system has discovered the patterns for the various combinations of the different parts of the body and the environment variables; then only the environment and selected parts of the human body are controlled, the system will generate the motion for the rest. The advantages of the framework are (1) the motion is fairly realistic since it is based on examples. (2) different parts of the human body may act less dependently; e.g. the top of the body might react to other environmental changes other than synchronizing with the bottom of the body. In section 2, we will introduce the theoretical basis for this motion generation system. In section 3, we will give the implementation suggestion from its realization in a computer game engine we developed. In section 4, we will discuss how the research fits into the large context of automatic motion synthesis in the computer game engine which includes both global and local animations.

2 The Humanoid Motion Generation Framework

2.1 Motivation from the Simulation-Theory

Motivation of the proposed motion generation framework came from the introspection that human beings are capable of generating first-person and third-person simulations in its brain. This capability leads us to one of the famous hypothesis concerning the human brain, which is called the Simulation-Theory (ST). The following two examples illustrate some of its basic ideas. Example one: suppose you are now reaching out your hand for a cup of coffee, why would you do this action? The hypothesis explains that you reach out your hand because you have simulated this action in your brain slightly before you perform it. And you reach out your hand instead of using it to move the mouse or stroke the keyboard (which might probably be the next actions if you are using a computer at the same time), possibly because your attention is then on the feeling of thirst and the cup of coffee. This does not mean that your brain was not simulating moving the mouse or stroking the keyboard at that time. Instead, these actions might also being simulated then, but they were neither performed nor emerged in the conscious brain because they were not magnified due to a selection process of the attention. Example two: suppose you are driving a car when there appear some people about to cross the road, how do you decide whether to brake your car or not. The hypothesis tells that your brain is continuously simulating the observed motion of the people ahead of time so that the attention of danger will not be signaled unless the simulation leads to a similar dangerous situation stored in the mind. And because simulation takes place before it actually happens, the redistribution of attention during simulation decides whether you brake or not. I.e. If danger is signaled, the shifted attention will magnify (select) the brake simulation and braking will be performed.

For years, researchers have suspected that the binding task (mind and brain) is accomplished by nerve cells in distinct areas of the brain communicating between themselves by oscillating in phase (40 hertz) -- like two different chorus lines kicking to the same beat even though they're dancing in different theatres. These oscillations have been detected in everything from the olfactory bulb of rabbits to the visual cortex of cats and even conscious humans. IBM, Birmingham and Saint Mary's researchers believe they have explained not only how the oscillations come about but also how the oscillatory rhythm is communicated from one area of the

brain to another. These two findings are critical to understanding how the complex electric signals of large numbers of nerve cells generate awareness and perhaps even consciousness.

This coherency or synchronization among different groups of nerve cells is a sign of concurrent simulations in the brain. According to the ST theory, by performing simulations in a similar manner, it is possible to generate new motions for an animation system. In our proposed motion generation framework, a learning and simulation algorithm is used to generate the motion for the characters. We first combine the animation variables and all the related environment variables which form the configuration space called DIM (dimension) space. Each variable or DIM in the configuration space is a function of time *f (t), t<=Tc*. Any subset of *these* DIMs and a segment of time [*T1, T2*] can be regarded as a simulation of these synchronized DIMs of length (*T2-T1*). An attention mechanism is used to keep track of and select the most relevant simulation(s) for each DIM. The value of *f (T+ ⊿T)* for every DIM is computed from its selected simulations unless overridden by an external user (supervisor or environmental inputs). Details will be given in section 2.3

2.2 Review of the Simulation-Theory

The Simulation-Theory (ST) is originally a theory about how the human brain generates visual imagery. A recent description of this theory can be found in papers by Hesslow [8]. I will further develop this theory or hypothesis to make it easy to understand from a computer point of view.

ST states that human imagination and visual/auditory perceptions are in essence the same thing in our conscious mind, and that they are both the input and output of the unconscious mind which does the work of recognition, memorization and deduction. The cycle of imagination and the subconscious forms mostly a closed loop when we are asleep, and a biased loop (by what we perceive) when we are awake. See **Figure 1**.

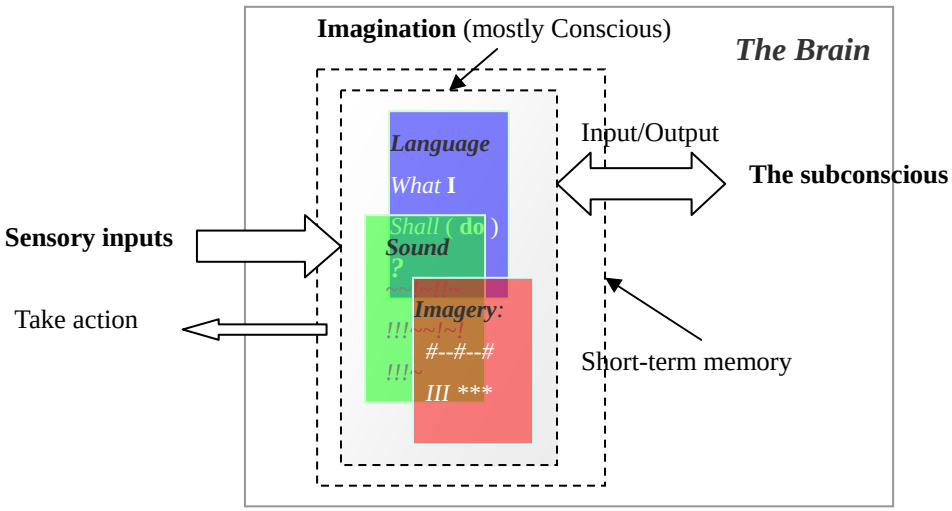

Figure 1. Human Brain: The Imagery-subconscious loop

Metaphorically speaking, the Imagination can be thought of as a multimedia, virtual reality "theatre" [9], where stories about the body and the self are played out. These stories,

- are influenced by the present situation according to perception,
- elicit the subconscious activities accordingly,
- and thereby influence the decisions taken by action.

The following parts are the key ingredients in the ST theory. They are “dimensions of simulation”, “concurrent simulation”, “imagery-subconscious loop” and “attention and memory”.

(1) **dimensions of simulation.** Any real world situation can be mathematically dissected into infinite number of functions $f_n(t)$ with each evolving over the same time variable *t*. Given *N* samplings of these functions, we obtain an *N*-dimensional simulation of a real world situation, i.e. $\{f_n(t) \mid n = 1, 2, ..., N; t \in [T_0, +\infty]\}$. Generally speaking, the similarity between the simulated situation and the real world situation increases as the number of dimension *N* increases. So long as the number of dimensions is high enough, the two very different systems (i.e. the simulated system and real world system) are analogous to each other. We will propose later that the number of concurrent simulations in the human brain is not one, but many; and they are simulated with varying degrees of dimensions or similarities to the real world. Simulation with the highest dimensions is most analogous to the real world situation and becomes our conscious or internal perceptions. The brain mechanism controlling the selection of simulation dimensions is called “Attention”.

(2) **concurrent simulation.** At any given time, our brain is simulating thousands of visual/auditory/etc. situations concurrently. These simulations are all of high interest or relevancy to the current state of the brain. But only a selected few are enlarged (i.e. their number of dimensions is increased) to be perceptible in our inner world (consciousness).

(3) **imagery-subconscious loop.** This part has been described in many other literatures [8, 9]. It just states that simulation in the human brain can evolve on its own, without interacting with the real world. This is achieved by feeding the result of a simulation to the input of itself, hence forming a loop between imagery and subconscious in the brain.

(4) **attention and memory.** Attention selects only a limited mount of imagery at any given time, despite there might be millions of other stories that are being played (simulated) in the mind at the same time. It is the constant selection of our attention that constitutes what we perceive as a continuous consciousness. Attention replays a previously unmagnified simulation or memory clip in the same sequential order as it was generated some time ago, therefore reinforced it in the memory; it signifies the importance of such imagery by bringing it to our internal perceptions, which in turn, makes it easier to affect subsequent imagery generation and selection of attention.

2.3 Mathematical and Architectural Formulation

To make things more precise, we now give a more formal formulation of the motion synthesis problem for animated characters. Please see section 3 for the implementation of the formulation.

2.3.1 Formulation of the Physical World

The notation adopted here is loosely based on the conventions used in [10].

1. The 3D environment in which the characters move is denoted by *W* (commonly called the workspace in robotics), and is modeled as the Euclidean space R^3 (*R* is the set of real numbers).
2. All local environmental variables are denoted by $\mathbf{e} \in E$, a vector of *m* real numbers, specifying the environmental or emotional states relevant to a certain character.
3. A character or agent is called *A*. If there are several characters, they are called *Ai (i = 1, 2, …)*
4. Each character *A* is a collection of p links *Lj (j = 1, … , p)* organized in a kinematic hierarchy with Cartesian frames *Fj* attached to each link.
5. A configuration or pose of a character is denoted by the set *P = {T1, T2, … ,Tp}* of *p* relative transformations for each of the links *Lj* as defined by the frame *Fj* relative to its parent link's frame. The base or root link transformation T1 is defined relative to some world Cartesian frame *Fworld*.
6. Let *n* denote the number of generalized coordinates or degrees of freedom (*DOFs*) of *A*. Note that *n* is in general not equal to p. For example, a simplified human arm may consist of three links (upper arm, forearm, hand) and three joints (shoulder, elbow, wrist) but have seven *DOFs (p = 3, n = 7)*. Here, the shoulder and the wrist are typically modeled as having three rotational *DOFs* each, and the elbow as having one rotational *DOF*, yielding a total of seven *DOFs*.
7. A configuration of a character is denoted by $\mathbf{q} \in C$, a vector of *n* real numbers, specifying values for each of the generalized coordinates.
8. Let *C* be the configuration space or *C-space* of the character *A*. *C* is a space of dimension *n*.
9. Let *Forward(**q**)* be a forward kinematics function mapping values of *q* to a particular pose *P*. *Forward(**q**)* can be used to compute the global transformation *Gj* of a given link frame *Fj* relative to the world frame *Fworld*.
10. Let *Inverse(P)* be a set of inverse kinematics (IK) algorithms which maps a given global transformation *Gj* for a link frame *Fj* to a set *Q* of values for **q**. Each configuration $\mathbf{q} \in Q$, represents a valid inverse kinematic solution (*Forward(**q**)* positions the link *Lj*, such that the frame *Fj* has a global transformation of *Gj* relative to Fworld). Note that the set *Q* may possibly be infinite, or the empty set (no valid solutions exist).

2.3.2 Formulation of the Motion Control System

Any motion for the character (including the changes in its local environment) will trace out a curve (i.e. a path) in this multi-dimensional space as illustrated in **Figure 2**. The curve is normally a piecewise continuous function of time (i.e. a trajectory). Conceptually, the fundamental goal of the motion synthesis strategy is to generate trajectories in the configuration space, so that they are in harmony (e.g. trajectories of the environment variables match the desired trajectories of the animation variables.). The character will either act out its planned motion or the one dictated by a supervisor. In whichever ways, the character will observe its final actions and always consider them to be in harmony. The continuous observation provides the basis for the generation of its future harmonious motions.

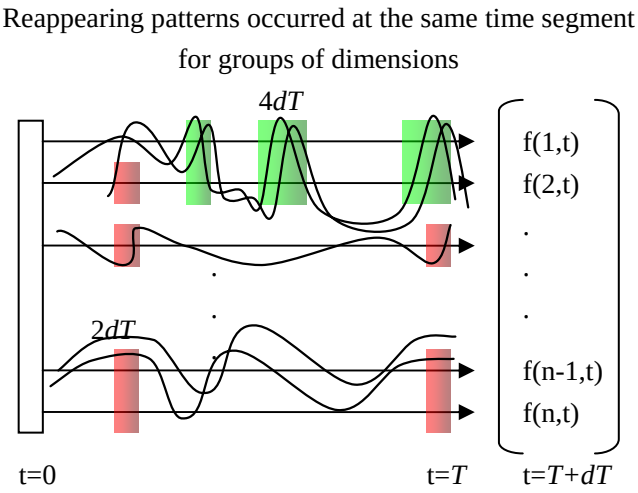

Figure 2. Motions for a character trace out a time-parameterized curve in the multi dimensional space.

1. A snapshot of a motion at time *t* is denoted by **f** (t) = $<f_i(t)> \in D$, where $D = E \bigcup C$. **f** (t) is a vector of $N = (m+n)$ real numbers. The vector space *D* is called DIM (dimension) space which combines the configuration of the environment variables and that of the animation variables.

2. A *simulation* is defined as *sim*(**n**, *t1*, *t2*) = $\{f_{n_i}(t), t \in [t1,t2] | n_i \in \mathbf{n}\}$, where

   $$\mathbf{n} \subseteq \{i \,|\, 1 \le i \le N, i \in \mathrm{N}\}$$

3. Let **ALL** *(t)* be a set of *N* functions of time *t*, also defined by *sim*({1,2,…,*N*}, *0*, *t*).

   Let *f (i, t) :=* $f_i(t)$ denote the *i*th dimension or componet of **ALL** *(t)*.

2.3.3 Motion Generation Algorithm

In this section, we will propose the motion generation algorithm. A motion system is denoted by **motion** (*Tc*) := < *Tc*, **ALL** *(t)*, **attention**, **relation**, **User** >. Let **MoGen** denote the motion generation algorithm so that **motion** ($t+\Delta t$) = **MoGen (motion** (*t*)).

Tc is the current time, where *Tc*>=0.
ALL *(t)* has the same definition as in Section 2.3.2, where 0<=*t*<= *Tc*.
attention is a function mapping each component of **ALL** *(t)* to a value att **(t)**, it is also written as

att (t):= $\{att_i(t) | \forall f_i(t) \in \mathbf{ALL}(t)(att_i(t)=attention(f_i(t)))\}$

relation is an *N* dimensional matrix. The value **R**i,j = **relation**(i,j) denotes the degree of relativity between *f (i, t)* and *f (j, t)* for all *t*.
User is an external supervisor. It has the right to override values in **ALL** *(t)* for all *t*. **User** is an unknown process to the motion system, but is executed at the end of every time slice.

motion ($t+\Delta t$) is computed from **motion** (*t*), using the following iterative process.

(1) $f_i\ (T_c+\triangle T) = f_i\ (T+\triangle T)$, where T satisfies $att_i(T) = \max\{att_i(t) \mid 0 \le t \le (T_c-\triangle T)\}$ if no external input from the **User**; otherwise $f_i\ (T_c+\triangle T) = User_i(T_c+\triangle T)$. For $t \in (T_c, T_c+\triangle T)$, an interpolation function can be used such as

$f_i\ (t) = (f_i\ (T_c+\triangle T) - f_i\ (T_c))/\triangle T \times (t-T_c) + f_i\ (T_c)$. After this step, **ALL** *(t)* of **motion** (*Tc*+ $\triangle t$) can be obtained.

(2) Compute **att** (*t*) of **motion** (*Tc* + $\triangle t$) from **att** (*t*) of **motion** (*Tc*) and **ALL** *(t)* of **motion** (*Tc*+ $\triangle t$), using Simulation-theory(ST rules). See **Figure 3** (**att** (t) denotes **att** (*t*) of **motion** (t)*).*

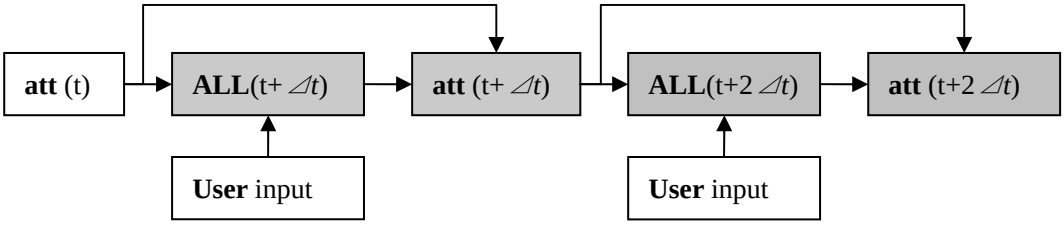

Figure 3. Iterative process of the motion generation algorithm

Simulation-theory (ST rules) is the center of the algorithm. It is used to calculate the redistribution of **attention** for **ALL** (*t*). To express the ST rules more clearly, the encoding of **ALL** (*t*) is given and a few other related helper functions are also defined.

ALL (*t*) is encoded using the following method. For every component f(i,t) of **ALL** (*t*), partition f(i,t) into minimum number of segments such that f(i,t) is monotonic in each segment.

$$f_i(t) = \sum_{k=0}^{num-1} q_k(t-T_k), \text{where } q_k(t) = f_i(t+T_k) \text{ if } 0 \le t \le T_{k+1} \text{ or } q_k(t) = 0 \text{ otherwise}.\ \{T_k\}$$

is the set of bending points in f (*t*), where T0=0, Tnum=f(Tc+ $\triangle t$). $\{[T_k, T_{k+1}] \mid k=0,1,...,num-1\}$ is the set of all segments in the time domain.

For every component y=f(i,t) of **ALL** (*t*), define its inverse functions as:

i = q(t, y), where **e**.y = f(**i**, q(t, y)) and **e** is a unit vector with the same dimension of **i**. We can use this function to get the index of components of **ALL** (*t*) which has the value of *y* at time *t*.

t = g(i, y), where **e**.y = f(i, g(i, y)) and **e** is a unit vector with the same dimension of **t**. We can use this function to get the list of time when the *i*th component of **ALL** (*t*) has the value *y*.

ALL (*t*) is stored in the form of **t** = g(i, y), since it will be used most frequently in our algorithm. g(i, y) can also be rewritten as below, where ToVector({Xi}) := [X1,…,Xn].

$$g(i, y) = \text{ToVector}(\{t \mid \forall k \le num(t=q_k^{-1}(y) \text{ iff } (q_k^{-1}(0)-y)\bullet(y-q_k^{-1}(T_{k+1}) \ge 0))\})$$

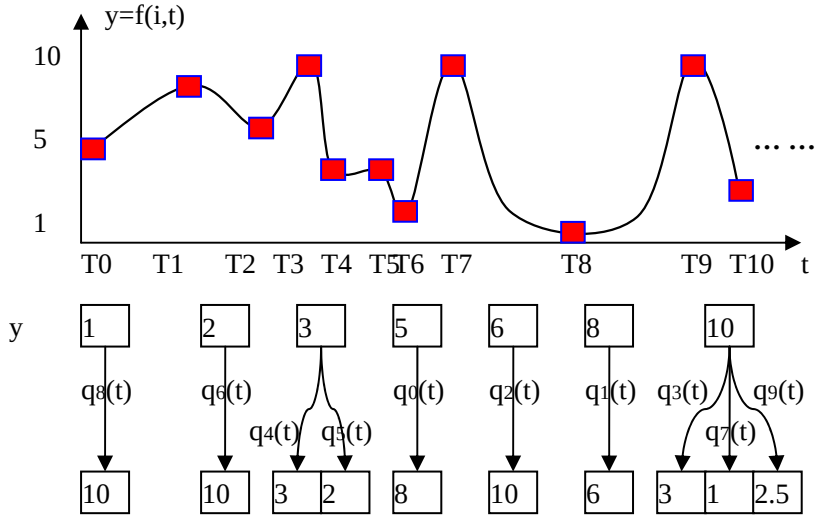

Figure 4. Data presentation of g (i,y)

Figure 4 shows the data presentation of g (i,y) (*i* is some fixed value.). E.g. in the above curve, g(i,10)=[3,7,9]. When implementing this data presentation, each $q_i(t)$ can be further compressed using parameterized analytical equations, when a certain tolerance of error is allowed. In other words, some sub curve $q_i(t)$ can be combined, if they are similar. An integer counter C_i is associated with each curve $q_i(t)$, which will be increased if our algorithm hits on the sub curve. Some rules are used to delete the sub curves from g(i,y) whose counter value is comparatively small in order to keep the size of g(i,y) constant when the length of curve increases ($t \to +\infty$).

Now we are ready to give the ST rules which are expressed in the form:
att (t+⊿*t*) = ST3(ST2 (ST1(**ALL**(t))), **att**(t)), where ST1, ST2, ST3 are three rules used to modify the **attention** for the motion generation system. Each rule is given below with their explanations.

ST rule 1

$$ST1(f_i(t)) := \sum_{k=0}^{maxlen} \left(\frac{1}{(k+1)\sqrt{2\pi}\sigma} e^{-(t-g(i,f_i(T_c+(1-k)\triangle T)))^2/(2\sigma^2)}\right)$$

ST1 searches each f(i,t) of **ALL**(*t*) for the longest sequence of matching sub curves in the form $f_i(t)$, $t \in [T_c+\triangle T-\triangle Len, T_c+\triangle T]$,$(\triangle Len=\triangle T \times maxlen)$. ST1 returns a value for each *t* $(t < T_c+\triangle T-\triangle Len)$, indicating how the curves near *t* resembles the latest occurring curves. The colored boxes in **Figure 2** show the effect of ST1. The curves in these boxes resemble the latest occurring curves (its color also shows the degree of similarity). **Figure 5** shows the curve of ST1 for a sample f(t).

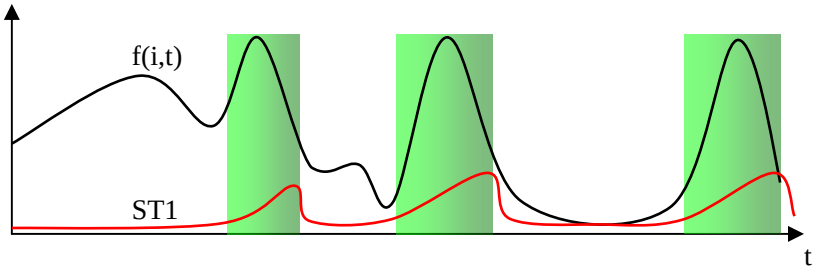

Figure 5. Sample curve of ST1

ST rule 2

$ST2(\mathbf{x}(t)) = \sum_{i=0}^{N-1} x_i(t)$, where **x** is an *N* dimensional vector of functions

ST2 is the rule used to find synchronizing positions in a set of functions of *t*. In the algorithm, the result from ST1 is used as input of ST2, i.e. ST2 (ST1 (**ALL** (t))). It does so by simply summarizing the components of **x**(t). The result of ST2 is just a single function of time, showing the places in the time domain where the reappearing patterns of **x**(t) occurs. **Figure 6** shows the sample curve of ST2. The apex of its convexes denotes the center of synchronization position. And its magnitude is roughly denotes the number of synchronization component. In ST theory, maximums in the curve of ST2 are the places in the time domain, where different kinds of simulations may occur. The **attention** mechanism will select among these simulations to decide the next output for the motion generation system.

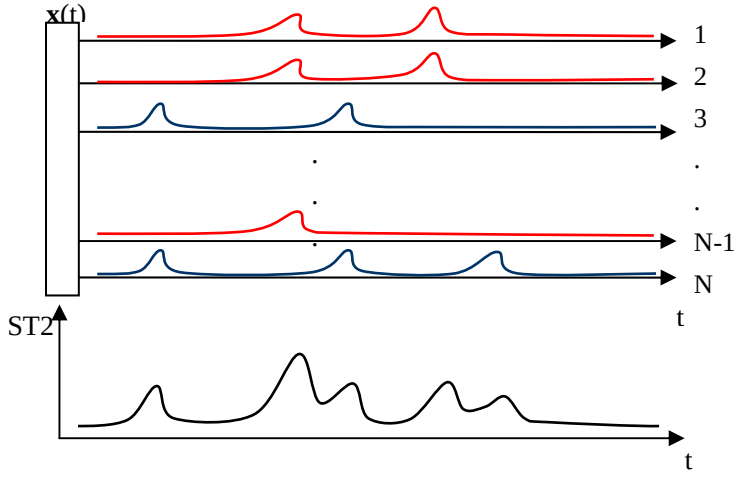

Figure 6. Sample curve of ST2

ST rule 3

$ST3(x(t), att(t)) = \text{attention}(\text{divide}([\text{sort}\{T_i\}\text{by}\int_{T_i}^{T_i+\Delta T} x(t)dt], att(t)))$, where {Ti} are the maximums in the curve of x(t). (x(t) is usually the result of ST2.)

ST3 is the rule used to generation the **attention**. It first sorts {Ti} by the area of a small region after Ti along the x(t) curve. Then it use a *divide* function to partition all f(i,t) into $\{\{f_{a_{i,j}}(t)\}_j\}_i$ according to **att**(t) and the sorted maximum points. In **Figure 6**, a possible division would be

{{f(1,t), f(2,t), f(N-1,t)}, {f(3,t), f(N,t)},…}. Each group is synchronized by one of the time in {Ti}. After deciding this division, the new **att**(t) are generated, so that the attention for components in the *i*th group is raised to a value at least higher than all other positions in the time domain of that component. Components in the same group form a specific *simulation* which can be used to generate subsequent motions for these components. A general degradation is performed in **att**(t), so that the total amount of attention is a constant. The division function is affected by both the result from ST2 and the last distribution of attentions. Because the *divide* function is flexible, different motion generation systems might adopt different *divide* criterion and no details of it will be given here. However, as a general rule, we should try to minimize the total number of groups (simulations) during the division. The **relation** matrix can be used in the *divide* process to minimize the number of groups. E.g. components of high relevancy tend to be included in the same group.

2.3.4 A Discussion of Performance

When implementing the algorithm, a lot of things can be simplified. The data structures for ST1, ST2 and ST3 can be in the form of list of key points, since the information in their curves are few compared to the length of their time domains.

The space complexity of the algorithm is O(*Fea*N*). *Fea* is the average number of reoccurring patterns (simulations). *N* is the number of dimensions in **ALL**(*t*).

The time complexity of ST1, ST2, ST3 can be O(log(*Fea*)* *Fea**N), O(N), O(N*N*N) respectively. So the total time complexity of the algorithm is $O(\mathrm{Fea}\times\log(\mathrm{Fea})\times\mathrm{N}+\mathrm{N}^3)$. In a typical case of humanoid animation system, *Fea* is usually several hundreds and *N* is usually round 50. It is possible to use the motion generation algorithm for real-time character animation.

3 Implementation

In this section, we will give the implementation suggestion from its realization in a simple computer game engine we developed.

To be continued.

4 Application in the Game Engine

In this section, we will discuss how the research fits into the large context of automatic motion synthesis in the computer game engine which includes both global and local animations.

4.1 Introduction to Automatic Motion Synthesis

First of all, the proposed motion generation algorithm is not the replacement to all local animation solutions in a computer game engine, due to performance and other requirements such as precision and ease of development. It is up to the game developers to decide when and which characters are going to use a certain kind of animation engine.

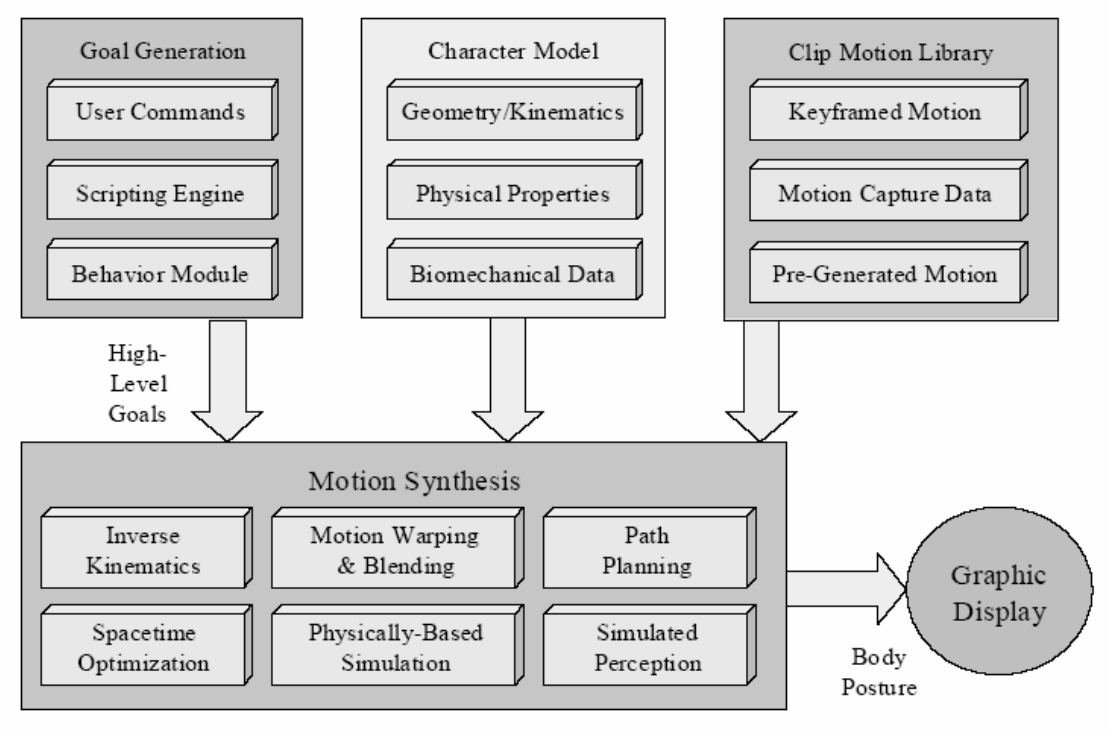

Figure 7. Resources available for motion synthesis [1]

Their presentation and implementations can be found in my other literatures [3, 4]. Generally speaking, the new motion generation system is used in the game engine only for the main character(s) when it is in the view frustum. Now, we are going to see how to divide the control variables relevant to the character between the global motion planning and local motion animation.

Figure 8 (left) shows a typical character. For fast motion synthesis, we want to only consider planning for the degrees of freedom that affect the overall global motion of the links of a character (i.e. the root joint DOFs). For the joint hierarchies we used, each of the 6 degrees of freedom of the Hip joint falls into this category. The idea is to compute motion plans for these DOFs (which we shall refer to as active DOFs or active joints), while the remaining joints (which we refer to as passive DOFs or passive joints) are either held fixed or controlled by a local animation algorithm. For our navigation strategy, passive joints are animated either by the new motion generation system or by cycling through a clip motion locomotion gait. However, passive joints could also potentially be driven by the active joints with their motion computed using a simple mathematical relation or even a sophisticated physically-based simulation (e.g. the motion of a character's hair in response to the gross motions of the head. In these cases, I call them add-on animations.).

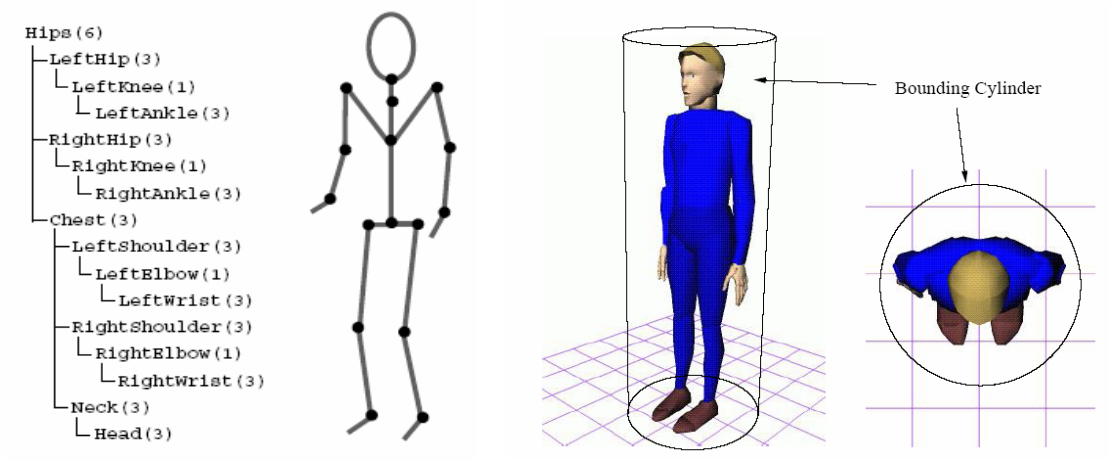

Figure 8. The major joints in the kinematic hierarchy used for locomotion are shown, along with the number of DOF for each joint.

Moreover, we can further reduce the dimensionality (and hence the efficiency) of the planning phase by considering only a subset of the active joints when possible. For instance, if we assume that the character navigates on a surface, we can omit one of the active translational DOFs (the height of the Hip joint above the surface), as well as two of the active rotational DOFs (by assuming the figure's main axis remains aligned with the normal to the surface). Suppose all of the passive DOFs and the omitted active DOFs are driven by a controller that plays back a simple locomotion gait derived from motion capture data. By approximating the character by a suitable bounding volume (such as a cylinder), that bounds the extremes of the character's motion during a single cycle of the locomotion gait, we have effectively reduced the navigation planning problem to the 3-dimensional problem of planning the motion for an oriented disc moving on a plane. **Figure 8**(right) shows one of the characters used in our experiments along with its bounding cylinder.

In case we do not want to generate location from recorded motion clips, we can use the new motion generation algorithm. The control variables that should be taken from the global animation are the velocities for all the DOFs of the character’s root joint (Hips). Behavior commands can also form a special dimension in the algorithm. The environmental variables such as a sound source or terrain features relative to the character or its emotions can also be included as different dimensions in the algorithm. All these additional dimensions are overridden by the **User** (as is called in our algorithm) and used to for the motion generation system to produce motions for the rest of the dimensions.

4.2 Animation System in the Game Engine

In ParaEngine (the name of our computer game engine), several global timers are used to synchronize engine modules that need to be timed. **Figure 9** shows a circuitry of such modules running under normal state. The darker the color of the module is, the higher the frequency of its evaluation. The local animation system is part of the rendering engine and is evaluated at a very high frequency. In other words, by the time the animation system is called, (1) user or script commands have already been interpreted into valid actions of the character, (2) global path-planning has already been processed and the character knows its preferred and valid position,

orientation and velocity in order to reach its destination on the map. Hence the task of the animation system in the game engine is to choose a right pose for the character and render it so that its animation matches its global velocity and action descriptions. It is also possible to let the animation system deal with all the add-on animations (hair, cloth, smoke, etc) and slight variations on the terrain.

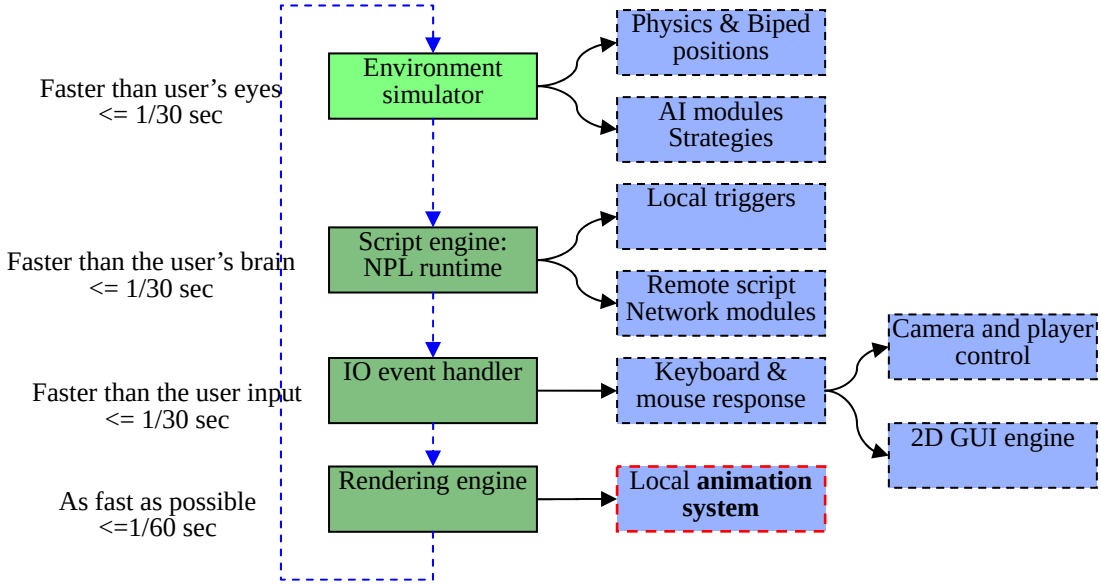

Figure 9. Engine modules and their evaluation frequency

In the game engine, there are two skeleton animation options, which can be used simultaneously. One is using pre-recorded motion clips; another is using the proposed motion generation algorithm. **Figure 10** shows a screen shot of many animated characters in a game scene constructed by our game engine.

Figure 10. Screen shots from Parallel World game

4.3 Discussion of the Animation System

One of the disadvantages of using the proposed motion generation algorithm is that it needs to maintain a relatively large amount of data for each of its animation instance. Yet, with pre-recorded motion clips, only the animation index of two adjacent animation sequences and a

frame counter for the current animation need to be kept for each instance. Hence, for large quantity of animation instances, motion clips based animation is still the preferred choice. However, when there is only a single character to control, as in many role playing games, the proposed motion generation system will produce more diversified and realistic animation. In reality, we can mix these two approaches even on the same game character. For example, when a character first came into the camera view, we can decide which animation system to use and use it until the character is out of the view.

To be continued

5 Conclusion

The fundamental challenges in computer graphics and animation lie primarily in having to deal with complex geometric, kinematic, and physical models. The development of better software tools has advanced the state of the art in the modeling and rendering of these models. However, software tools for the automatic generation of motion are still relatively scarce. In particular, techniques are needed to animate both autonomous and user-controlled human figures naturally and realistically in response to high-level task commands. Motivated by the Simulation-theory, this paper proposes a new motion generation system for real-time character animation. By using the proposed algorithm, the generated motion is fairly realistic and different parts of the character may act less dependently. Another advantage is that it no longer needs to manually build and maintain a large database of motion clips; instead, the algorithm will automatically learn and reapply known patterns according to the states of the character and its local environment.

The proposed algorithm is actually a general learning algorithm, which can be applied to a number of learning tasks. Human animation is an ideal test bed for the proposed algorithm.

Reference:

[1] Kuffner, J. J., Autonomous Agents for Real-Time Animation. Ph.D. thesis, Stanford University, 1999.

[2] Ying Liu, Interactive Reach Planning for Animated Characters using Hardware Acceleration. Ph.D. thesis, U. Penn, 2003.

[3] Xizhi Li. “Using Neural Parallel Language in Distributed Game World Composing”. In the Proceedings of IEEE Distributed Framework of Multimedia Applications 2005. (to be published)

[4] Xizhi Li. “ParaEngine: A Game Engine Framework for Distributed Internet Games”. (ready to submit)

[5] M. Girard and A Maciejewski. Computational modeling for the computer animation of legged figures. In Proc. of SIGGRAPH '85, 1985.

[6] R. Boulic, D. Thalmann, and N. Magnenat-Thalmann. A global human walking model with real time kinematic personification. The Visual Computer, 6(6), December 1990.

[7] S.-K. Chung and J.K. Hahn. Animation of human walking in virutal environments. In In Proceedings of CA '99 : IEEE International Conference on Computer Animation., pages 4{15, Geneva, Switzerland, May 1999.

[8] Hesslow, G. (2002) “Conscious thought as simulation of behaviour and perception.” Trends in Cognitive Sciences, 6:242-247

[9] Murray Shanahan. The Imaginative Mind A Precis. Conference on Grand Challenges for Computing Research (gcconf 2004)

[10] J. C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, Boston, MA, 1991.