http://dx.doi.org/10.5392/IJoC.2011.7.3.019

Fast Motion Synthesis of Massive Number of Quadruped Animals

Mankyu Sung
ETRI
161 KaJung-dong, Yusong-gu, Daejeon, KOREA

ABSTRACT

This paper presents a fast and practical motion synthesis algorithm for massive number of quadruped animals. The algorithm constructs so called speed maps that contain a set of same style motions but different speed from a single cyclic motion by using IK(Inverse Kinematics) solver. Then, those speed maps are connected each other to form a motion graph. At run time, given a point trajectory that obtained from user specification or simulators, the algorithm retrieves proper speed motions from the graph, and modifies and stitches them together to create a long seamless motion in real time. Since our algorithm mainly targets on the massive quadruped animal motions, the motion graph create wide variety of different size of characters for each trajectory and automatically adjusted synthesized motions without causing artifact such as foot skating. The performance of algorithm is verified through several experiments

Keywords: *Crowd Simulation, Motion Synthesis, Quadruped Animals*

1. INTRODUCTION

A large number of motion synthesis methods have been proposed by the computer animation research community in recent years. The primal reason for the high popularity of motion synthesis techniques corresponds with the wide spread of 3D computer games and wide use of computer graphics-based special effects in feature films or movies. However, most of the researches have focused on synthesizing motions for only human-like characters. Although the motions of human characters are still an animator's major concern most of the time, many non-human-like characters play important roles in current movies and animation films. For example, in movies such as "The Chronicles of Narnia" or "The Golden Compass" lots of animals support the main characters or are sometimes themselves the main characters. In general, animal motions are usually made by the traditional key frame method. Even though this method has an advantage in fine control over motion, it requires intensive labor, an artistic sense, and a significant amount of time for creating animation, which is a big hurdle for fast production. In particular, if we want to animate a massive number of animals, the problem gets even worse. Alternatively, for human motions, several data-driven methods are proposed to meet the realistic animation requirements as well as to satisfy user-specified controls. For example, graph-based methods [1], [2], [3] and statistical model-based methods [4], [5] have their own advantages and disadvantages for practical use. But essentially, those methods are tested only for human characters and are quite limited unless there is a large amount of motion data available.

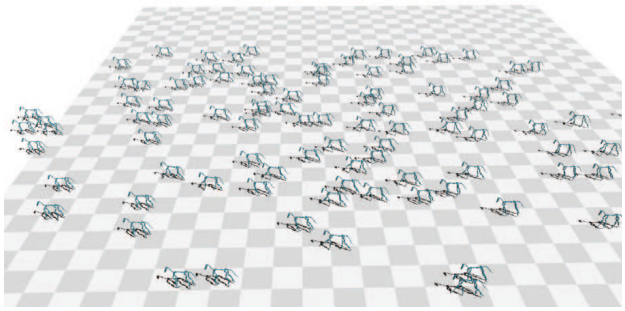

Fig. 1. 200 horses in movement : our algorithm can synthesize natural motions for a lot of quadruped animals.

This paper proposes a fast locomotion synthesizing method, shown in Fig. 1, for a large number of quadruped animals. This method is basically data-driven that uses motion capture data. However, it focuses on developing techniques that exploit an existing small number of motions extensively. This is an inevitable choice because capturing animal motion is very hard and quite expensive. Our method, on the other hand, requires only a single cyclic motion of four different quadruped gait types (walking, trotting, cantering and galloping) and transition motions between them. Then, the algorithm artificially creates a new set of the same type of motions with wide variation of different speeds, which we call a speed map. In order to obtain different motion speeds from the original input motion, the speed map motions analyze the gait pattern and apply the

* *Corresponding author. E-mail : mksung@etri.re.kr*
Manuscript received Jun 30, 2011 ; accepted Aug.09, 2011

specialized Inverse Kinematics(IK) algorithm to have different gait lengths and limb configurations. In constructing a speed map, the original styles of motion, especially the order of foot planting on the ground, are maintained because it characterizes the specific gait types of quadruped animals.

The various speedy motions in the speed map allow characters to transit motions smoothly from normal moving to fast moving, and vice versa. This feature is very important in real applications because training animals to obtain very specific speed motions is almost impossible, although most of the games and movies require very specific speed motions. These maps, constructed through the procedure above, are then connected with each other to form a well-known motion graph structure where four hub nodes represent four different locomotion types and edges represent speed map motions [1], [2], [3]. The construction of a speed map and motion graph is done as preprocessing steps. At run time, the user inputs a guideline point trajectory for each character. This trajectory is created by a user control or by outside simulators. The trajectory represents the root joint positions of a character over time. Thus, the time duration between the points represents a frame time, which means the closer the distance between two adjacent points, the slower the character moves. Given the trajectories, the algorithm puts an arbitrary size of animal characters on a trajectory and then starts to traverse the motion graph to retrieve proper speed motions out of the graph. These motions are warped to make the character follow the curve of the trajectory exactly. Finally, the retrieved motions are then smoothly stitched together to make long motions.

Our method has the following contributions over previous methods. First, one of the major disadvantages of motion graph based motion synthesis is that it requires a big corpus of motions to be used in real applications. Our method, on the other hand, does not require many motion data because we artificially synthesize many different motions and warp them to make them move on a randomly shaped curve. Second, we extend the existing foot skating clean-up algorithm for human characters [6] to four-legged animal motion data. In this process, all important characteristics of the original motion are maintained. Third, our algorithm is able to synthesize a large number of animal characters with a wide variety of different sizes at the same time. The input guide line trajectory is totally independent of specific characters. Therefore, the proposed method allows adding any size of character. The motion synthesis takes the character size as a parameter and synthesizes scaled motions accordingly

2. RELATED WORK

Traditionally, animal motions for feature films have been created through elaborate a key-frame animation technique [7]. Fine control over movement is the major reason for this dominant use of this method, although animators require so many reference videos and creativity to figure out their movement [7]. A lot of robotics researchers have been studying quadruped robots for several decades [8], [9]. However, these researches do not directly reflect the high realistic animal motions because their focus is on creating stable robots that can move on wide a range of different situations.

For realistic animation of animals, several researches have studied on how to capture real motions. Because of a high difficulty in capturing wild animals or dynamic human motions, they use live video sequences such as documentary film to reconstruct the 3D model of animals, or incorporate on existing motion capture database [10], [11] or copying and modifying motion from human motion capture data [22]. However, their methods do not create totally new motions unless there are live videos of the new motions. Also, only relatively simple motions are able to be captured. A physical-simulation method is another way to generate quadruped motions. Raivert *et al* designed a control system that activates or deactivates actuators to represent some particular behaviors for legged characters [12]. Torkos *et al* proposed a trajectory-based optimization technique for synthesizing motions for quadruped animals [13]. In their approach, users are required to input footprint locations, their timing, and stylistic hint for motions, the system then applies physics and optimization to estimate the body posture of the quadruped. Their approach is similar to our method in this paper in that both use footprint location to change the speed of motion, but we use motion capture data as the input data instead. Therefore, we do not need any physics or high computation load in the optimization process. Most recently, Coros *et al* proposes a physical model for quadruped animal that synthesize motions such as walk, trot, pace and canter [21]. Although their method can create a wide variety of motions automatically, the quality of motion does not meet the motion capture data.

Wampler *et al* proposed an optimization-based animal locomotion algorithm [14]. In their method, they took the shape of an animal and its motions as a component of continuous optimization framework. In order to animate physically realistic motions, they parameterize the radius and length of limbs so that they are adjusted for the given gait. Although their synthesized motions are physically realistic, they ignore the complexity of the non-mechanical structure of real animal legs, which sometimes does not convey the details of real animals.

For human motions, motion blending has been an efficient tool to create parameterized motions [15], [16]. However, one of the significant artifacts of motion blending is that it does not guarantee to satisfy the important kinematic constraint, a so-called foot plant, which requires some part of a foot to remain stationary on the ground. An explicit solver for the foot skating problem has been introduced in [6] by Lucas *et al*. We extend their bipedal IK solver to quadruped animals to find the limb configuration of animals when constructing the speed map. In this process, the beat pattern, which means the order of four feet contacting with the ground, is maintained. This pattern is important to characterize a different locomotion style. The motion graph technology has been drawing a lot of attention in the computer graphics community because of its simplicity and implementation ease [3], [1], [2]. However, one of the main

drawbacks is that it needs a big corpus of motion data for good connectivity in real applications. Zhao *et al* proposed a simple method that uses motion blending to create a new set of motions with similar poses as the original motion set for constructing the motion graph [17]. Our method is similar to

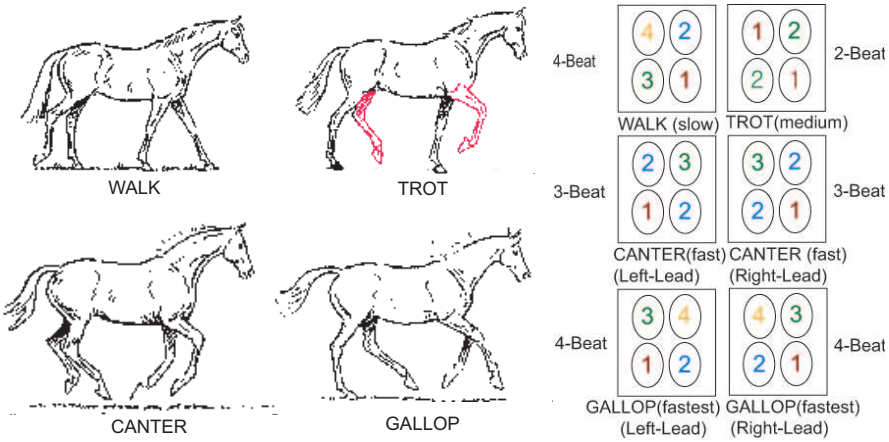

Fig. 2. Left: Four typical gait types of a quadruped animal (horse). Right: Beat patterns of four gait types *(equusite.com)*.

their method in that it also creates many motions out of a small set of input motions. However, we use a speed-parameterized IK technique instead of motion blending, which is required for synthesizing massive animal locomotion.

3. SPEED MAP CONSTUCTION

Because our goal is to synthesize motions for a large number of quadruped animals, we need a wide range of different speed motions beforehand. These motions, called a speed map, become input data in synthesizing motions at run time when a dynamic change of speed motions is required. In general, quadruped animals have only a few gait patterns that they use frequently, although there are many possible foot fall patterns [18]. Four typical gait patterns are shown in Fig. 2. Depending on the gait-patterns, the orders of footing on the ground are different, as shown in Fig. 2, and their speed is also changed. For example, the walk motion has a four-beat pattern with the slowest speed where the left hind leg is put on the ground first, and then the right fore leg is placed next, with the left hind leg and the left fore-leg placed later. On the other hand, the trot motion has a two beat gait with a faster speed than walking in which diagonal foot plants are always the same time stamps. The speed of motions, s, is defined as follows throughout this paper.

$$s = \frac{l}{n} \qquad (1)$$

where l is the length of the root joint trajectory and n is the number of frames.

Our basic strategy is to input a single cyclic motion for each gait type and automatically enrich it through parameterizing its speed as shown in Fig. 3. Under the definition of motion speed, to obtain a different speed of the original motion, we have two choices: adjusting the number of frames (n) or adjusting the length of the root joint trajectory (l). Since adjusting the number of frames is not what we want, we fix the number of frames and adjust the length of root joint trajectory. Specifically, let's say that the original input motion has speed, s, then we increment or decrement this speed by a predefined δ. Then, the new length becomes $l' = (s \pm \delta)l$ and $s = s \pm \delta$.

In this process, we have to make sure that two important features of the original motion are kept. First, all foot skating artifacts should be removed. Foot skating happens when the root joint position, which is responsible for the global position of the character, is not properly coordinated with the limb configuration. Therefore, simply forcing a change in the length of root trajectory causes a significant foot skating of the character. To remove the foot skating, we need to figure out the global position of end-effector joint at the contact frame and its time duration. Then, the IK solver finds proper limb configurations for that. Second, the gait beat pattern of the original motion as shown in Fig. 2, should be maintained. Even after we relocate the foot position for changing the speed, the order of foot plants should be preserved. In order to do that, we manually annotate the foot plant frames (f_c) that have a time interval, and the associated joint (j_c) of the input motions by the order of time, and then feed them into the construction step as a constraint. This constraint should be satisfied in creating the speed map.

The speed maps are constructed through the following order.

1. **Relocation of root joint position**: Let us say that $d_{i,j}$ denotes the 2D projected distance between root joints at frames i and j. We compute the $d_{f,f-1}$ for all adjacent frames. Then, given the new length of the motion trajectory, l', the amount of adjustment for $d_{f,f-1}$ is l'/f. This value is multiplied on $d_{f,f-1}$ to compute a new distance $d'_{f,f-1}$. Suppose that the global position of j joint at frame f is $P_j(f)$ where root joint has 0 in j. Then, the new root position, $P'_0(f)$, can be simply calculated as $P'_0(f) = P'_0(f-1) + {}'d_{f,f-1}, f \geq 1$. Note that if s is bigger than1, it creates a faster motion, whereas if s is smaller than 1, it generates a slower motion. Also, for the very first frame, $P'_0(0)$ equals $P_0(0)$.

2. **Finding the joint position of end effectors**: The next step is

to find the proper position of end effector joints, which is the joint contacting with the ground, that minimizes the foot skating. For this, from the constraint of the input motion indicating foot plant frames, f_c, and end effector joint, J_c, we first compute the average position $\dot{P}_{jc}(f_c)$ of J_c during the f_c of the original input motion. This is the original foot plant positions before relocating the root joint, so, we need to relocate the foot plant position as well. For this, we compute

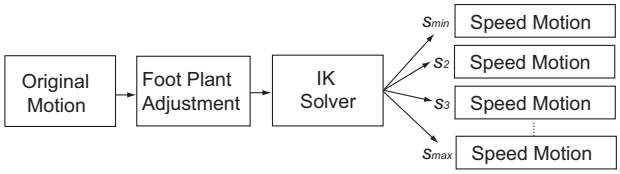

Fig. 3. Speed map construction process

the scalar value distance, d_c, which is the difference between $P'_0(f_c)$ and $P_0(f_c)$. This is the magnitude of direction vector, V, that needs to be added to the original $\dot{P}_{jc}(f_c)$. The only remaining problem is to find V. In our approach, we set the V by computing a vector between two adjacent foot prints of the same foot. Figure4 illustrates this process. The new foot plant position $P'_{jc}(f_c)$is computed by $P'_{jc}(f_c)= V+ \dot{P}_{jc}(f_c)$.

3. **IK solving**: Given $P'_{jc}(f_c)$, we apply the IK solver to find a limb configuration. To maintain the order of beat pattern of the original motion, we apply the IK by the order of fc. The IK solver we use is similar to the one that Lucas *et al* proposed in [6] with some extension for meeting the constraints above. We briefly overview our IK solver here with emphasis on the extensions that we made in this section. More detailed information can be found in [6].

Figure 5 shows four steps for IK processing. At the first step, the algorithm computes the hinge angle analytically. Concise derivation of the equation for computing this angle can be found [19]. One drawback of the original algorithm for computing the hinge angle is that it assumes that the plane of rotation of the hinge is defined by the thigh and shin. Most of the time, this hinge axis does not cause any problems, but when the original limb is almost fully stretched, which is the case for fast motions, slight changes of thigh and shin cause an inconsistent change of sign of hinge axis over the frames, which produces visually unpleasant hinge angle popping. This is because the hinge axis is computed by the cross product of two vectors, $\overrightarrow{P_rP_h}$ and $\overrightarrow{P_{jc}P_h}$ where P_r, P_h, and P_{jc} represent the global position of a limb root, hinge, and end effector, respectively. The cross product generates a near to zero magnitude vector when three positions are almost on a straight line, which is not a desirable axis. To solve this problem, our hinge axis is defined by two vectors, $\overrightarrow{P_rP_h}$ and $\overrightarrow{P_tP_h}$ instead where P_t is a position of the toe whenever the original definition of the hinge axis has a smaller magnitude than the predefined threshold. At the second step, transform the limb toward the target by finding the smallest amount of rotation that makes the vectors $\overrightarrow{P_rP_{jc}}$ and faces the same direction with $\overrightarrow{P_r\acute{P}_{jc}}$. At the third step, properly rotate the end effector orientation so that it does not penetrate the ground. At the final step, stretch the limb joint length unless it meets the target.

In the construction of a speed map for each gait type, we decrease speed parameter s from maximum speed s_{max} to s_{min} by δ. As a result, we construct roughly around 60 speed maps for each gait type. In this process, we intentionally overlap the speed range of two different gait types so that two speed motions from different gait types can have the same speed as shown in Fig.6. For example, a fast walking motion might have the same speed as a slow trotting motion. This feature is a big advantage for massive character animation where variation on motions among a crowd is a critical issue.

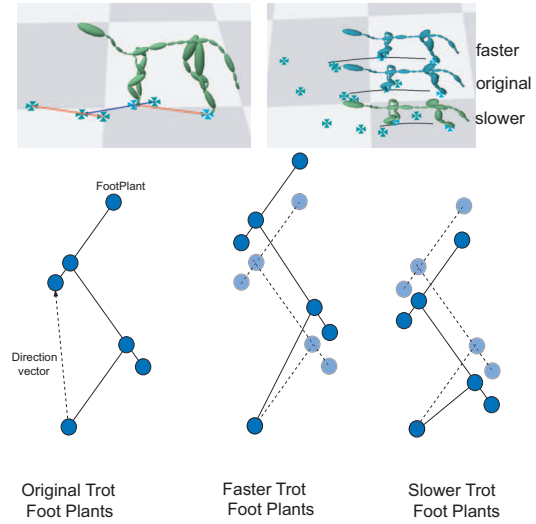

Fig. 4. Foot plant adjustment for changing the speed of trot motions. Blue circles represent the foot plants

4. BUILDING MOTION GRAPH

Constructed speed maps are connected with each other through a motion graph [3], [1], [2]. Fig. 7 shows a motion graph for quadruped animals (we only show a couple of edges per speed map for simplicity). In building the motion graph, transition motions between speed maps are also required for smooth motion change between different gait types. Like an original motion graph, an edge of the motion graph represents a typical motion and nodes represent common poses. The difference is that all edges in each walk, trot, canter and gallop node come from the speed map rather than the original input motions. When traversing the graph, we assume that all of the characters are starting at the standing node. As they speed up, they move up to the walk node, and then move to the trot node, canter node, and gallop node, sequentially. When they need to slow down, they follow the reverse order.

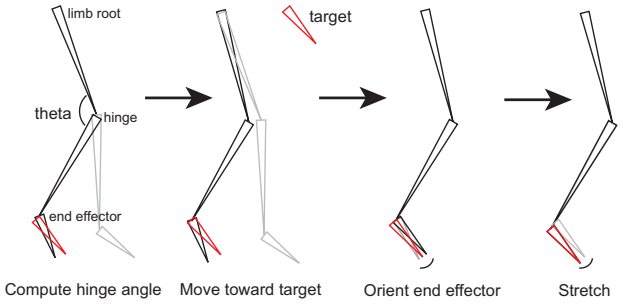

Fig. 5: IK process for finding limb configuration

5. GUIDELINE TRAJECTORY

Our approach requires the user to input a trajectory per character. This trajectory, $T_i(f)$, is a list of 3D points representing a guideline for root joint position of the ith character over time. The reason that we separate the simulation of movement of characters that would be stored as a trajectory from detailed motion synthesis of individual characters is that aggregate behaviors of herds or crowds are very important in massive character animations. An overall formation change of crowds over time needs to be simulated and edited frequently until the best scene comes up. Therefore, it is not desirable to synthesize detailed motions for every character whenever

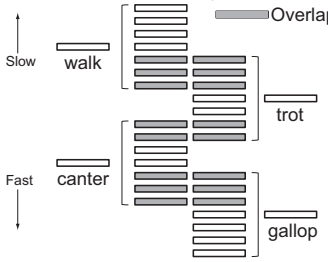

Fig. 6. The ranges of speed in speed map for four gait types

simulating the movement of herds or crowds. Our approach is to simulate the collision-free movement of the overall characters first, and then wear the detailed motions on them next.

If the number of characters is not so large, then we can create the trajectories in a manual manner, using a mouse drag to capture the trajectory of characters on a virtual 3D environment. However, if the number of characters is too big, then manual capturing is impossible. In this case, we can use other crowd or flocking simulation software such as Massive software or Boids model. In the case of using commercial software, we can make a simple script utility that stores the positions of characters over time as a text file.

1. Smoothing: One assumption that we make over the trajectories is that they should have smooth curvatures and not contain any sharp turns. The point in a trajectory represents a rough position of a character in the environment. Thus, even in our smoothness assumption, if there is a discontinuity or noise in the trajectory, synthesized motion does have quite noticeable popping either. To remove this, we filtered $T_i(f)$ with a low-pass filter. The low-pass filter can be a general Gaussian filter, which is widely used in image processing.

2. Re-sampling: We have to make sure that the speed of a trajectory is within the speed range of the speed map. Otherwise, we adjust the trajectory by the re-sampling process. The speed range of the speed map can be easily found simply by checking the fastest speed motion of the gallop speed map, s'_{max}, provided that the speed maps are constructed in such a way that their speed ranges are overlapped. Note that the slowest speed map is always zero because we assume that characters always start from a standing pose. As a result, the speed range of the speed map becomes $0 \le s \le s'_{max}$. Given this speed range of the speed map, we need to check the speed range of the trajectory for comparison. To check the speed range of the trajectory, we first segment the entire trajectory $T_i(f)$ by m points, which is the average number of frames of input walk, trot, canter, and gallop motion. As a result, we get n segments S_i.

$$\begin{aligned} T_i(f) &= \{S_0, S_1, S_2, ..., S_n\} \\ S_i &= \{P_0, P_1, P_2, ..., P_f\} \\ \text{where } P_i \in \mathbb{R}^3 &\text{ and } (i-1)\cdot m \le f \le i \cdot m \end{aligned}$$

For each segment S_i, we compute the speed $s(S_i)$. After computing the speed over all segments and sorting them, we obtain the speed range $s^i_{min} \le s(T_i) \le s^i_{max}$ for the trajectory. By checking the speed ranges of all trajectories in this way, we can get the speed range, $s_{min} \le s(T_*) \le s_{max}$, for all trajectories. Then, the two speed ranges, (s_{min}, s_{max}),

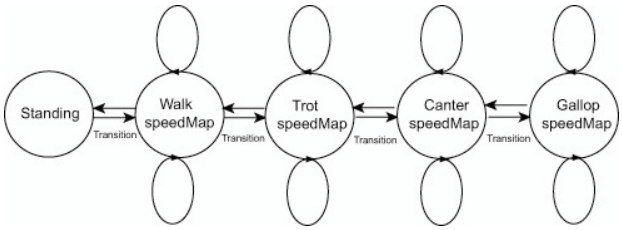

Fig. 7: A motion graph : four speed maps for quadruped animals are connected with each other with transition motion

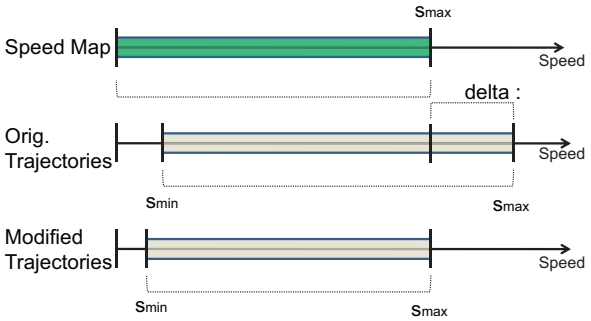

Figure 8: Speed range adjustment of a trajectory for a given speed map

which is the speed range of the trajectory, and $(0, \acute{s}_{max})$, which is the range of the speed map, are checked to see whether the speed range of a trajectory is within the range of the speed map. If the speed of the trajectory is out of the range

of the speed map, then our

algorithm is not able to synthesize motions for the speed. To solve this problem, we re-sample the trajectory to change its speed. Unlike the case of speed map, where we change the length of the trajectory to adjust the speed, we change the number of points of the trajectory by fixing the length of the trajectory. The reason for this is that the geometrical characteristics of the trajectories such as length are a product of a crowd simulator's high level path planning and interaction among characters. Thus, it should be preserved.

The sampling rate *r* is calculated as follows:

$$r = \frac{\acute{s}_{max}}{s_{max}}$$

Because of re-sampling, the range of trajectory turns to $r \cdot s_{min} \leq s(T_*) \leq r \cdot s_{max}$ as shown in Fig.8. For example, if *r* is 0.5, then we make all characters move at twice slower speed.

One important point is that we must apply the sampling rate *r* to the all trajectories. If we apply the new sampling only on a few characters, which would make them move faster or slower, the global formation of characters would be broken.

6. MOTION SYNTHESIS

Without loss of generality, we assume that animal characters are at the beginning of their trajectory. Under this condition, the motion synthesis step needs to create a long motion that makes the character move along the trajectory while satisfying the speed. Our basic strategy is to traverse the motion graph, shown in Fig.7, and find the proper edges. Those edges, which correspond to a piece of speed map motion, are then blended together to get an exact speed motion. The blended motions are then placed one after another along the trajectory. Motion warping is necessary to bend the original motion trajectory to match the trajectory. Transition motion is also placed when it needs to traverse across different gait types. We describe the details of each step in this section.

Since our goal is to synthesize motions for a lot of quadruped animal characters, variation on the character size makes the scene more interesting. To achieve the randomness on the character size, we multiply the scale factor *h*, which is a real value that is always bigger than minimum h_{min} but smaller than a threshold maximum h_{max}, to all joint offset vectors to scale up or down the character size. We then randomly distribute a scale-adjusted character to all the trajectories.

After deciding the character size for each trajectory, the algorithm starts to synthesize motions from the start of the trajectory to the end. Motion synthesizing steps are illustrated in Fig.9.

The algorithm first computes the speed of trajectory, *s*, over *m* points of the trajectory. Then, using speed parameter *s*, it traverses the motion graph to find the two closest edges of the graph that have the speed *s*. One edge (M_1) should be faster than *s*, and the other edge (M_2) should be slower than *s*. Note that because we overlapped the speed ranges of speed map, there might be multiple styles of motion that contain the speed *s*. In this case, randomly select one style. The reason for selecting the two best matched edges is that the speed map is constructed in a discrete speed space. Thus, in most cases, there is no motion that perfectly matches speed parameter *s*. We resolve this through interpolation. Suppose that M_1 has speed $s(M_1)$ and M_2 has speed $s(M_2)$. Then, we can obtain the speed motion *M* with exact speed *s* by interpolating $M_1$and M_2 with weight value α.

$$M = h((1-\alpha)M_1 + (\alpha)M_2)$$

where $\alpha = \frac{s(M_1)-s(M_2)}{s-s(M_2)}$ and *h* is the scale factor.

The next step is to *warp* the motion *M* so that its motion trajectory has the same shape of the $T_i(f)$ where $n \leq f \leq m$. Figure 10 shows the warping process of two speed motions in a row. The warping process includes two specific jobs. First, based on the tangent of input trajectory, which is computed as $\theta = \arctan(d_x, d_z)$, where $d = P_{i+1} - P_i$ we rotate the orientation of the root joint by Θ. The root position $P_0(f)$ is also repositioned using 2D transform matrix ***T***, composed with rotation Θ and translation $P_0(f)$-$P_0(f-1)$.

This process is similar to the motion path planning algorithm [20]. Second, the warped motion $\acute{M}$ is exactly located on T_i using a 2D transform matrix that makes $P_0(f)$ into P_n, as can be seen in the top image of Fig.10.

As a final step, newly warped motion ($\acute{M}$) is smoothly stitched with existing synthesized motions ($T(M)$) through motion blending. For this purpose, we need margin frames (around 10 frames) in the front and back of the input cycle motions, as can be seen in the bottom of Fig.10.

The motion synthesis steps are iterated until it meets the end point of the trajectory.

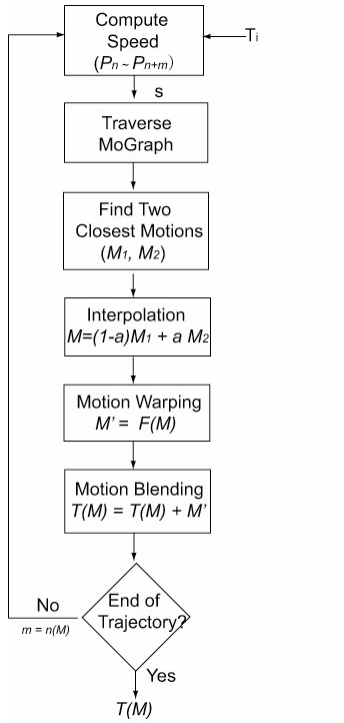

Fig. 9. Motion Synthesis Flow Diagram

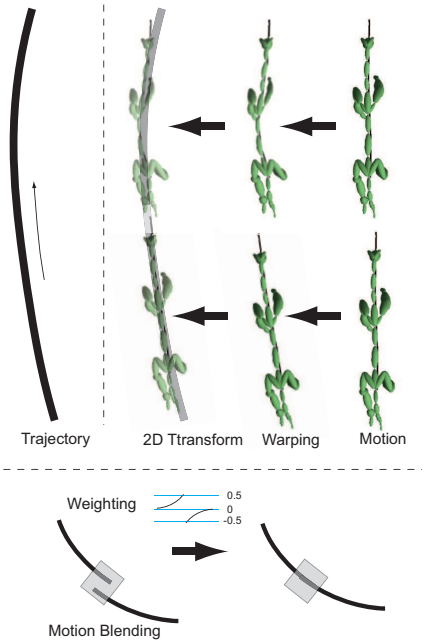

Fig. 10. Top: Motion Warping Bottom: Motion Blending

7. EXPERIMENTS

To validate our algorithm, we have performed experiments using animal motions. However, lack of available animal motion capture data allows us to apply our algorithm on quite a limited number of animal types. We bought commercially available animal motion capture data from a motion capture studio (The3DStudio.com).

We use the animal motions of a horse and dog for our experiments. These motions are all sampled at 60Mhz. For the horse, only 12 motion clips are used; four of them are a single cycle of *walk, trot, canter, and gallop* motion for constructing the speed map, and the rest of them are transition motions such as *static2walk*, *walk2static*, *walk2trot*, *trot2walk*, *trot2canter*, *canter2trot*, *canter2gallop* and *gallop2canter*. For the dog, on the other hand, 9 motions are used because gallop motion is not available.

All experiments were tested on an Intel Xeon 3.20GHz processor PC with a graphics acceleration card.

Table1 shows detailed information about the speed maps. Note that the speed range of each gait type is overlapped and all speed map motions have the same number of frames of input motion.

The first experiment is for testing whether our algorithm can synthesize motions for any arbitrary curved path with various speed changes. For this, we ask users to drag a mouse on the 3D virtual floor to represent the movement of a character, and use it as a input trajectory. As a result, our experiment verified that our algorithm makes the character follow the path exactly and synthesize various styles of motion depending on the speed change of the trajectory, as shown in Fig.11

The second experiment is for testing the performance of our algorithm. We use *OpenSteer* library to simulate a maximum of 500 characters (http://opensteer.sourceforge.net/). Specifically, we use the *Pedestrian* plug-in that makes agents follow a predefined control path. We put a small code in the simulation loop to capture around 500 frames of 3D position of characters over time and write out into a file. Figure 12 shows snapshots of the simulation rendered with Maya.

Two specific tests are performed based on these trajectories. First, as we increase the length of the trajectories from the starting point to the end, we compute the total synthesizing time of 500 characters. For comparison, we also test when we synthesize all motions on the fly without using speed map. In this case, to get the exact speed motion, we have to process all root trajectory adjustment and limb configuration changes at run time.

The result shows that the time cost increases almost linearly as a function of trajectory length. And, on-the-fly synthesis is almost twice slower than our speed map construction synthesis. Second, we calculate the average time for synthesizing a single frame as we increase the number of characters. Our result shows that it takes only an average of 0.008 seconds for synthesizing 480 frames for a character, which means that our algorithm can synthesize motions at realtime speed, as shown in Fig.13.

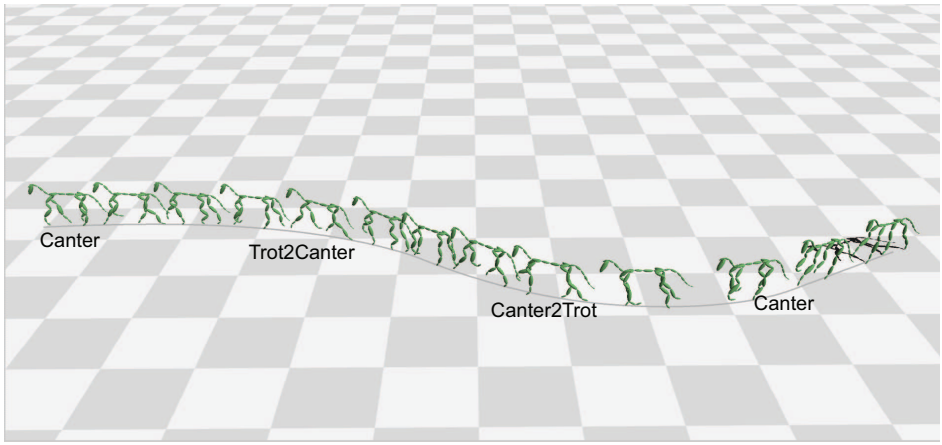

Fig.11. Motion synthesis result for a single character with various speed changes

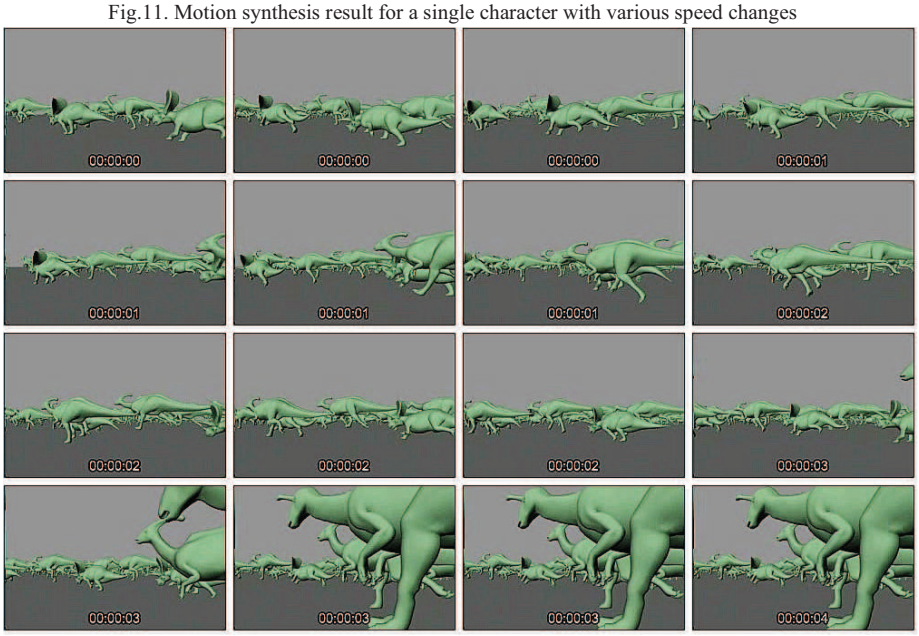

Fig.12. 300 dinosaurs running in herd (including biped dinosaurs)

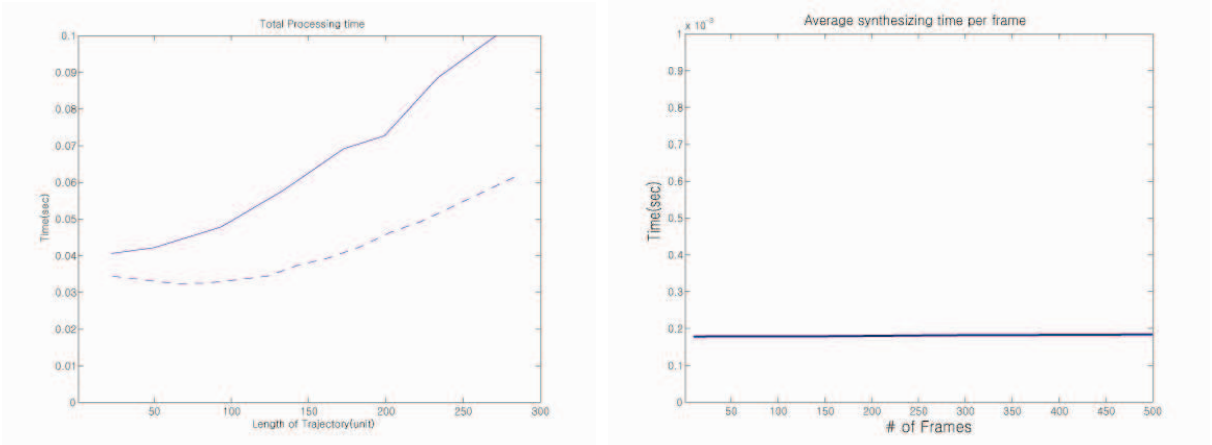

Fig.13. Left: The total processing time with increasing the length of the trajectory.(Dashed line is the synthesis speed using speed map, and solid line is the on-the-fly synthesis speed) Right: The average synthesizing time per frame, as a function of the number of characters

Table1. Information regarding speed map of four gait types of horse motion

Gait type	Time (sec)	#of motions	Speed range	#of frames
Walk	3.34	55	(0.08, 0.21)	82
Trot	2.86	55	(0.19, 0.39)	56
Canter	2.69	55	(0.36, 0.45)	51
gallop	2.45	55	(0.41, 0.49)	51

8. DISCUSSION AND CONCLUSION

In this paper, we introduce a fast and practical motion synthesis algorithm for large number quadruped animals. The core of this algorithm is to construct the speed map, which is a set of speed-adjusted motions made from a single cyclic input motion using a specialized IK solver. For quadruped animals, four speed maps (walk, trot, canter, and gallop) are constructed by the order of speed. Speed ranges of four speed maps are overlapped to present various styles of motions for a same speed parameter. Four speed maps are then connected as a motion graph along with transition motions. Given the trajectories representing the movement of characters, which are obtained from external software or a manual process, the algorithm synthesizes a long motion that makes the characters move along a curve exactly while satisfying speed constraints.

Although our experiments prove that this algorithm can synthesize a large number of characters in real time without causing significant artifacts, there are a couple of limitations.

First, we assume that input trajectories always have a smooth curve and no sharp turns. This is not a serious limitation, we believe, because quadruped animals show smooth turns most of the time except at special urgent events or after instructions from a trainer. Second, our motion synthesis algorithm is a purely kinematic method that does not consider the biophysical and ethological information of animals. In order to improve the motion quality, we need to employ useful information on animal behaviors and the physical characteristics of various animals that many veterinary and ethology scientists have researched.

REFERENCES

[1] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard,"Interactive control of avatars animated with human motion data," ACM Transactions on Graphics, vol.21, no. 3, 2002, pp. 491–500.

[2] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," ACM Transactions on Graphics, vol. 21, no. 3, 2002, pp. 483–490.

[3] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," ACM Transactions on Graphics, vol. 21, no. 3, 2002, pp. 473– 482.

[4] Y. Li, T. Wang, and H. Shum, "Motion texture: a two-level statistical model for character motion synthesis," ACM Transactions on Graphics. vol. 21, no. 3, 2002, pp. 465–472.

[5] M. Brand and A. Hertzmann, "Style machines," in SIGGRAPH'00: *Proc. of the 27th annual conference on Computer graphics and interactive techniques, New York, NY, USA,* 2000, pp. 183–192.

[6] L. Kovar, J. Schreiner, and M. Gleicher, "Footskate cleanup for motion capture editing," *in SCA '02: Proc. of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, New York, NY, USA,* 2002, pp. 97–104

[7] D. Wright, B. Westenhofer, J. Berney, and S. Farrar,"The visual effects of the chronicles of Narnia: the lion, the witch and the wardrobe," Comput. Entertain., vol. 4, no. 2, 2006, p. 4

[8] Y. Fukuoka, H. Kimura, and A. H. Cohen, "Adaptive dynamic walking of a quadruped robot on irregular terrain based on biological concepts," The International Journal of Robotics Research, vol. 22, no. 3–4, 2003, pp. 187–202.

[9] M. Fujita and H. Kitano, "Development of an autonomous quadruped robot for robot entertainment," Autonomous Robots, vol. 5, 1998, pp. 7–18.

[10] L. Favreau, L. Reveret, C. Depraz, and M. Cani, "Animal gaits from video: comparative studies," Graph. Models, vol. 68, no. 2, 2006, pp. 212–234.

[11] M. Park, M. Choi, Y. Shinagawa, and S. Shin, "Videoguided motion synthesis using example motions," ACM Transactions on Graphics., vol. 25, no. 4, 2006, pp. 1327–1359.

[12] M. H. Raibert and J. K. Hodgins, "Animation of dynamic legged locomotion," SIGGRAPH Comput. Graph., vol. 25, no. 4, 1991, pp. 349–358.

[13] N. Torkos and M. V. de Panne, "Footprint–based quadruped motion synthesis," *in Proc. Of Graphics Interface,* 1998, pp. 151–160.

[14] K. Wampler and Z. Popovi´c, "Optimal gait and form for animal locomotion," ACM Transactions on Graphics, vol. 28, no. 3, 2009, pp. 1–8.

[15] L. Kovar and M. Gleicher, "Flexible automatic motion blending with registration curves," *in Proc. of ACM SIGGRAPH/Eurographics symposium on Computer animation.* 2003, pp. 214–224

[16] S. Park, H. Shin, and S. Shin, "On-line locomotion generation based on motion blending," *in Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002, July 2002.*

[17] L. Zhao and A. Safonova, "Achieving good connectivity in motion graphs," Graph. Models, vol. 71, no. 4, 2009, pp. 139–152.

[18] J. J. Robilliard, T. Pfau, and A. M. Wilson, "Gait characterization and classification in horses," The Journal of Experimental Biology, vol. 210, 2007, pp. 187–197.

[19] J. Lee and S. Shin, "A hierarchical approach to interactive motion editing for human-like figures," *in SIGGRAPH '99: Proc. of the 26th annual conference on Computer graphics and interactive techniques, New York, NY, USA,* 1999, pp. 39–48.

[20] M. Gleicher, “Motion path editing,” *in Proc. Of 2001 ACM Symposium on Interactive 3D Graphics. ACM, Mar.*
[21] S. Coros, A. Karpathy, B. Jones, L. Reveret, M. van de Panne, “Locomotion Skills for Simulated Quadrupeds,” *in SIGGRAPH ’11: Proc. of the 38th annual conference on Computer graphics and interactive techniques, New York, NY, USA, 2011*
[22] K. Yamane, Y. Ariki, J. Hodgins, “Animating non-humanoid characters with human motion data,” in SCA *'10 Proc. of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation Eurographics Association Aire-la-Ville, Switzerland, Switzerland*

Mankyu Sung
He received the B.S and M.S in computer science Chung Nam National University, Korea in 1993, 1995 respectively and also received M.S. and Ph.D. in computer science from University of Wisconsin-Madison, USA in 2005. Since then, he has been with the Electronics and Telecommunication Research Institute (ETRI), Korea. His main research interests include computer graphics, computer animation and human-computer interaction.