Space-Time Warp Curve for Synthesizing Multi-character Motions

Mankyu Sung and Gyu Sang Choi

This paper introduces a new motion-synthesis technique for animating multiple characters. At a high level, we introduce a hub-sub-control-point scheme that automatically generates many different spline curves from a user scribble. Then, each spline curve becomes a trajectory along which a 3D character moves. Based on the given curves, our algorithm synthesizes motions using a cyclic motion. In this process, space-time warp curves, which are time-warp curves, are embedded in the 3D environment to control the speed of the motions. Since the space-time warp curve represents a trajectory over the time domain, it enables us to verify whether the trajectory causes any collisions between characters by simply checking whether two space-time warp curves intersect. In addition, it is possible to edit space-time warp curves at run time to change the speed of the characters. We use several experiments to demonstrate that the proposed algorithm can efficiently synthesize a group of character motions. Our method creates collision-avoiding trajectories ten times faster than those created manually.

Keywords: 3D motion synthesis, 3D crowd animation.

Manuscript received Sept. 20, 2016; revised May 11, 2017; accepted May 22, 2017. This work was supported by the National Research Foundation of Korea and funded by the Korean government (Grant No. 2015R1D1A1A010590).

Mankyu Sung (corresponding author, mksung@kmu.ac.kr) is with the Department of Game and Mobile Contents, Keimyung University, Daegu, Rep. of Korea.

Gyu Sang Choi (castchoi@ynu.ac.kr) is with the Department of Information and Communication Engineering, Yeungnam University, Gyeongsan, Rep. of Korea.

I. Introduction

During the last decade, crowd simulation has been a hot topic in the computer-animation research community. It is widely used in a variety of applications, for example, games or feature films. However, animating a number of characters still poses many technical hurdles that must be overcome. First, each individual must have realistic movements, although group behavior is considered as important as individual behavior. Second, characters must avoid collisions during simulations because they are visually annoying artifacts. However, it is not easy to check for collisions as the number of individuals increases. Third, we need an easy-to-use interface to specify the crowds' trajectories. As it is unimaginable to specify every trajectory manually, this needs to be accomplished automatically. Further, we would need to provide a method to adjust the trajectories afterward.

This paper proposes a new motion-synthesis method for crowd animations. At the high level, we propose a simple yet quite practical hub-control scheme for automatically specifying trajectories of individual characters. In this scheme, users are first required to input a single scribble in the virtual environment by dragging the mouse. Then, the trajectories are fitted with cubic B-spline curves, because the scribble may have undesirable sharp corners that are not appropriate for smooth motions. Users who wish to change the shape of the trajectory can adjust it by simply changing the positions of the control points of the spline curve. Then, from the initially fitted spline curve, our algorithm automatically creates n different curves through the hub-sub-control-point scheme.

In this scheme, hub-control points are usually control points from the initially fitted curve. Sub-control points are a set of control points maintained *inside* the hub-control points for automatically generating different curves. The hub and sub-control points are given offset distances to obtain different shapes from the original curve.

At the low level, our algorithm creates motions on the curves with a single cyclic motion-capture sequence. For synthesizing motions, we propose using a disparity map that indicates the orientation difference between the root joint and the curve tangent to improve motion quality. During this motion synthesis, space-time warp curves are automatically embedded in the environment.

The space-time warp curve is created based on the speed of a character. That is, the upward vector of the curve is the time domain, and the slope of the curve changes according to the motion speed. Users can observe whether the created trajectories cause any collisions by simply checking whether any space-time warp curves intersect. The time warping of the motions, which is responsible for changing the motion speed, is easily achieved by dragging the curve upward or downward with the mouse. We performed a series of experiments to validate our algorithms.

II. Related Work

A wide variety of approaches for simulating and editing crowd motions have been proposed. Classical approaches include particle systems or social forces [1], [2], which attempt to control crowds using simple dynamics. Rather than managing crowds as an aggregate of individuals, some approaches consider crowds as a continuum object [3]. Although these approaches may be useful for simulating the aggregate behavior, they do not consider the quality of individual motions.

To overcome this problem, motion-synthesis researchers have proposed several techniques for editing a group of character motions [4]–[6]. Their approaches are mainly inspired by mesh-deformation techniques that allow users to manipulate a particular vertex and deform the entire mesh interactively. If we imagine an individual in a crowd to be a vertex, the deformation technique is quite effective. However, although these techniques can produce natural motions using pre-captured motion clips, their interface is not easy to use because they have separate interfaces for controlling the spatial trajectory and timing control. Moreover, their approaches do not provide a method for controlling the finely detailed motions of individuals because the deformation affects the animation of the entire crowd.

Our proposed approach is similar to theirs in that we also synthesize individual motions using motion data. However, unlike their approach, we propose a scribble-based interface to specify the initial trajectory of the crowds and propose space-time warp curves that are embedded directly into the environment to control the spatial and timing changes of motion.

The scribble-based interface is quite useful because we can create a number of different trajectories with a single stroke. A similar scribble-based interface was introduced in [7]. However, their goal was to coordinate motions between multiple characters using a motion graph. Therefore, the scribble acts as a rough path for synthesized motions. A hard-constrained path cannot be easily satisfied because of the nature of the motion-graph-based motion-synthesizing algorithm.

For globally coordinated crowd movement, Barnett and others proposed the Reeb graph approach, based on a harmonic field for given starting and goal positions of individuals [8]. This approach produces sampled paths called “guidelines” from the globally coordinated graph. This approach is quite efficient for simulating group behavior; however, it has difficulty manipulating each trajectory in a highly detailed manner. Other researchers recently proposed a patch-based approach [9], which constructs environmental building blocks known as motion patches, which are annotated with motion data. The patches are then connected to build more-complex interactions. Our approach can be easily extended to accommodate this type of patch data structure because it can abstract and encapsulate a set of trajectories.

III. Algorithm

1. Path Creation through a Hub-Sub-Control Point Scheme

The proposed algorithm requires the user to input an initial path. Intuitively, this path represents a rough path along which the crowds move. We use a scribble interface that enables a user to directly draw a path on the 3D environment by dragging the mouse. We project 2D mouse positions onto 3D positions in the environment by simply shooting a ray from the 2D screen coordinate onto the 3D environment floor and then determine the 3D intersection positions. While the mouse is being dragged, we track the distance between the current position and the previous position. If the distance is shorter than a pre-defined distance threshold, it is discarded. This can minimize the number of 3D points needed to construct the initial path.

Suppose P^i, where $0 \leq i \leq n$, is a set of points representing the initial path. Starting with P^i, we fit the points with a cubic B-spline curve. Spline fitting has several advantages. First, it can transform the rough initial path into a smooth curve, which is required to obtain natural motions. Second, we can easily adjust the shape of the curve by changing the positions of the control points. However, since we aim to generate *many* trajectories for multiple characters, a single curve generated from the initial scribble is insufficient.

Our algorithm generates n different curves automatically by enabling each main control point to maintain a list of sub-control points. The sub-control points can be used to create another spline curve; however, their positions are constrained by the parent control point. Therefore, when a parent control point changes its position, its entire set of sub-control points changes accordingly because the distance between the parent control point and sub-control points remains constant.

The positions of the sub-control points should satisfy two requirements. First, the distance between each sub-control point must exceed the predefined minimum distance. Second, the distance between the parent control point and each sub-control point must maintain the predefined minimum distance. We refer to the parent control points as hub-control points. The hub-sub-control-point scheme is used to more easily generate multiple crowd trajectories.

In our scheme, we only allow the user to move the parent control point. The sub-control points are re-located automatically as the parent control point changes its position. Our algorithm simply maintains the initial 3D offset distances between the parent and sub-control point even after the parent point changes its position. This may change the trajectories' shape. However, we believe that it is acceptable for crowds because it allows more variation in the movements. If we want to keep the initial shapes of all the trajectories, even after changing the parent control point, we may need an optimization process to find optimal positions for the sub-control points. In this case, it is better to use a mesh-deformation technique; see [4], for example.

Figure 1 shows five different curves that were created from a single scribble. Note that since the sub-control points maintain their offsets from the hub-control points, their curves have different shapes.

In our approach, the B-spline fitting of a set of initial scribble points starts by deciding the number of hub-control points. The number of hub-control points is determined by the length of the initial scribble. The number of hub-control points H_n can be calculated as

$$H_n = \frac{L}{S}, \tag{1}$$

where L is the length of the scribble and S is the length of the trajectory of a single cyclic motion, which is input by the user. An excessive number of control points is difficult to maintain, even though it allows finely detailed curve changes. Too few control points are undesirable for the opposite reason. Our approach considers the length of the single cyclic motion to be used for the synthesizing motion. Thus, our aim is to place a control point whenever we repeat a cyclic motion, thereby minimizing unnatural motion.

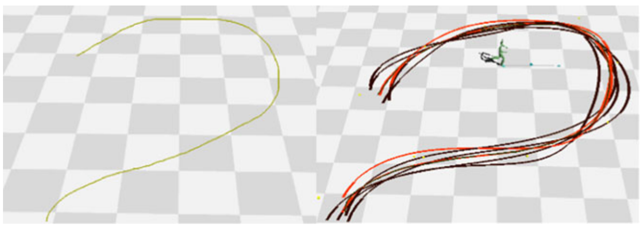

(a) (b)

Fig. 1. (a) Initial scribble and (b) set of curves generated by hub-control points.

The position of the cubic B-spline at parameter u can be obtained as [10]

$$p^u = \sum_{i=0} c_i \mathrm{B}_i(u), \tag{2}$$

where $\mathrm{B}_i(u)$ are the basis functions. For the cubic B-spline, the basis functions are

$$\begin{cases} B_0 = \frac{1}{6}u^3 \\ B_1 = \frac{1}{6}[1+3u+3u^2-3u^3] \\ B_2 = \frac{1}{6}[4-6u^2+3u^3] \\ B_3 = \frac{1}{6}[1-3u+3u^2-u^3] \end{cases}. \tag{3}$$

If we transform the above equation into matrix form, we obtain

$$P^u = [u^3 \quad u^2 \quad u \quad 1]\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} [c_i \quad c_{i+1} \quad c_{i+2} \quad c_{i+3}], \tag{4}$$

where $0 \le u \le H_n - 3$.

Given n scribble points P^i, where $0 \le i \le n$, and a cubic spline curve, fitting the cubic B-spline curve basically entails finding the positions of the hub-control points c_i The solution can be found in a least-squares manner by solving a linear equation formed from (4). This equation can be represented in the form $Ax = B$, where matrix x contains the control points, B is an $n \times 2$ matrix representing the scribble points, and matrix A is an $n \times H_n$ matrix that computes the right-hand side of (4) above, except

$$c = \sum_{i-1}^{n} (p^i - p^u)^2. \quad (5)$$

Equation (5) can then be solved using a popular decomposition method, such as the Cholesky decomposition [11].

Once we construct a cubic B-spline curve from the scribble points, we extend it by automatically adding sub-control points, which automatically generate a set of different curves. The offset distance between the original hub-control point and sub-control points is defined beforehand and can be set differently for each hub-control point. This benefits controllability because we can control the dispersion and scattering of the crowds.

Figures 2 and 3 show the relation between the distance constraint and its effect on the crowd trajectories. Note that we also enforce a constraint *between* the sub-control points, such that they also maintain the minimum-distance constraint. The simple pseudo-code for setting the positions of the sub-control points is given below. This aims to reduce the chances of different characters colliding with each other. The number of offsets is the same as the number of individuals.

```
//min_inter: minimum inter-control point
distance
//min_dist: minimum distance from the hub-
control points
WHILE(1)
    X=rand(); Y=rand();
    FOR I to N
      if dist(X,Y,Point[i])<min_inter &&
        dist(X,Y,CX,CY))<min_dist)
          Point[N]=Point(X,Y);
          N++;
          BREAK;
      END IF
    END FOR
END WHILE
```

The proposed hub-sub-control-point scheme has an advantage from a user-interface point of view. If we were to separately generate *independent* curves, the number of control points would be too many to handle. However, since our approach maintains offset constraints on the sub-control points, moving the hub-control points results in the related sub-control points automatically changing their positions.

2. Two-Step Motion Synthesis on the Curve

Given the curves, we need to synthesize motions such that the 3D characters move along the curves. Our

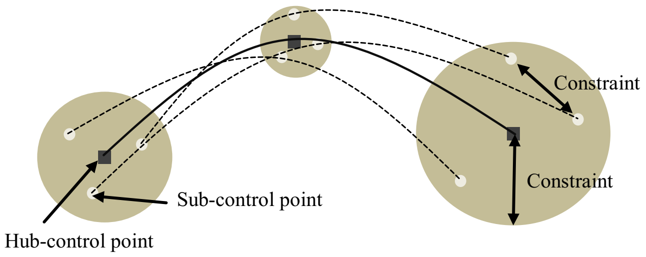


Fig. 2. Hub-control points and sub-control points.

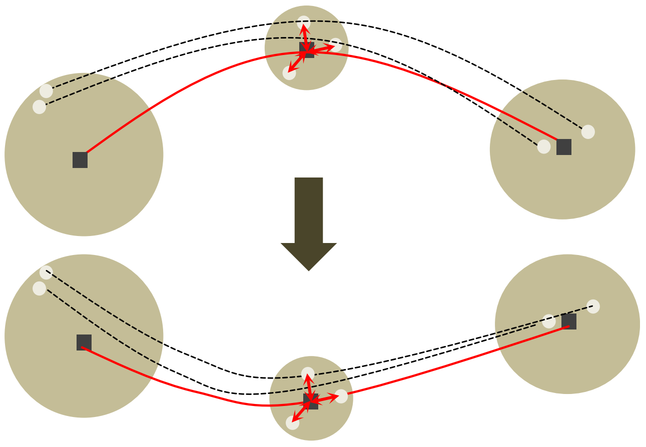


Fig. 3. Distance-constrained sub-control points.

algorithm repeatedly uses a single cyclic motion. A particular i frame of a cyclic motion consisting of m frames and k joints can be represented as a vector, as follows:

$$M^i = \{p_0^i, q_0^i, q_1^i, \ldots, q_k^i\}, p_i \in R^3, q_i \in S^3, 0 \le i \le m, \quad (6)$$

where p_0 is the root-joint position at frame i, which corresponds to the global position of the complete character, q_0 is its global orientation, and q_j is the local orientation of joint j. In most cases, the motion skeleton is constructed hierarchically and the root joint is the pelvis joint located at the center of the body.

Since the position and tangent of a particular parameter u can be analytically evaluated from the B-spline equation, the problem is reduced to finding u from the motion data. We use an approach similar to that proposed by Gleicher and others [12], who proposed an arc-length parameterization to find the parameter u based on the length of the arc. The arc-length between two adjacent frames can be calculated as $\|p_0^i - p_0^{i-1}\|_{\text{proj}}$, where $\|p\|_{\text{proj}}$ is the 2D projection of 3D point p onto the ground.

However, if we use their technique, the subtle nuances of the original motions are difficult to preserve when we change the root-joint orientation based only on the curve tangent. Thus, for our algorithm, we propose a two-step method to synthesize the motion for cubic B-spline curves. In the first step, we first fit the original cyclic motion with a cubic B-spline curve as we did for the scribble. Then, from the spline curve, we perform arc-length parameterization to find the parameters for all the frames. Once we find the parameters, we calculate the tangent angles for all the frames. We then construct a *disparity map* that shows the 3D-orientation difference between the tangent angle of the curve and the root-joint orientation q_0.

Mathematically, suppose a is the 2D tangent angle at the particular parameter u of the curve. Then, the disparity map $d(u)$ at parameter u can be calculated as

$$d(u) = q_u\left(\cos\left(\frac{a}{2}\right), 0, \sin\left(\frac{a}{2}\right), 0\right).q_0^{-1}, \qquad (7)$$

where $q(w, x, y, z)$ is the 3D quaternion representation.

Once we construct the disparity map, the algorithm synthesizes a series of frames as it increases parameter u from 0 to $H_n - 3$. Note that if the arc-length of the curve is longer than the original motion, we repeat the motion. The new orientation q'_u at a particular u on the cubic B-spline curve from the scribble can be computed as

$$q'_u = q_u.d(u). \qquad (8)$$

Figure 4 summarizes the two-step motion-synthesis algorithm.

3. Space-Time Warp Curve

Space-time warp curves are embedded in the 3D environment and represent the trajectory over time. Figure 5 shows a sample of the space-time warp curve. This curve consists of 3D points w^i whose x- and z-coordinates are the 2D projected positions on the ground from the root-joint position at every frame of the motion.

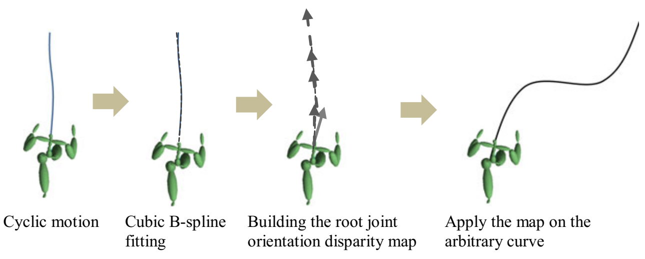

Cyclic motion
Cubic B-spline fitting
Building the root joint orientation disparity map
Apply the map on the arbitrary curve

Fig. 4. Two-level motion-synthesis algorithm.

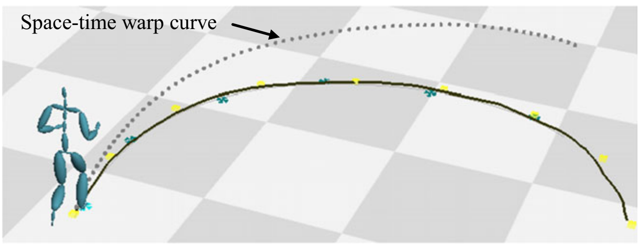


Fig. 5. Example space-time warp curve.

The y-coordinate represents particular times on the time domain and increases as a function of evolving time.

In our approach, the slope of the curve reflects the speed of the motion. If the motion occurs at a constant speed, the slope is also constant. The y-coordinate $w_y(i)$ at frame i can be calculated as

$$w_y(i) = \delta + \left[\frac{\|p_0^i - p_0^{i-1}\|_{\text{proj}}}{\Delta t}\right] s, \delta > 0, \qquad (9)$$

where δ is a constant, Δt is the time duration between two adjacent frames, and s is the scaling factor.

The purpose of the space-time warp curve is to control the timing of the motion and visualize the interaction among the crowds. Many other approaches, for example, the time-warp curve proposed by Kim and others [6], apply separate interfaces to change the motion speed. However, it is difficult to use these interfaces to change the speed at a particular moment in the environment because the timing control and positional control are separate.

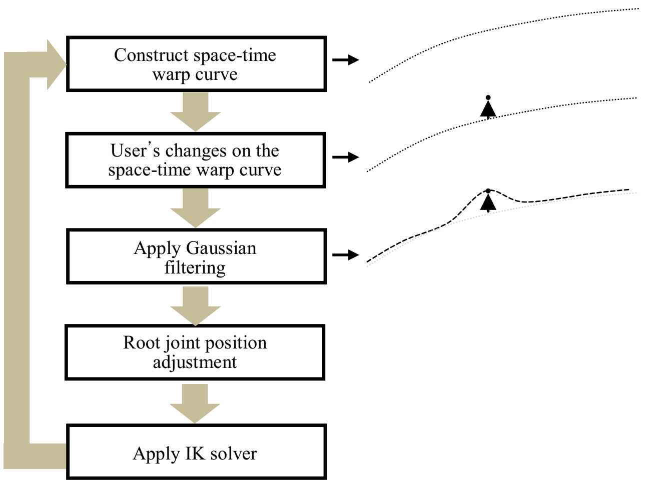
Our proposed algorithm allows the user to change the position of a particular point on the space-time warp curve by simply dragging it with the mouse. The position change is constrained only by the y-axis value. Then, from the above equation, the new position of the root joint is calculated from $w'_y(i)$ as

$$p_{0_{\text{proj}}}^i = p_{0_{\text{proj}}}^{i-1} + \left[w'_y(i) - \delta\right]\frac{\Delta t}{s}. \qquad (10)$$

The new root position may cause artifacts, such as foot-skating actions, as we were forced to change the root-joint position. We solve this problem by applying the foot-skating solver proposed by Kovar and others [13].

Their algorithm applies the specialized inverse-kinematic (IK) solver to adjust the knee and ankle orientation for given foot-plant positions. To apply the IK solver, we specified foot plants as tags in the motion data. These tags indicate the joints that should touch the ground during a particular range of frames. Our algorithm uses the XML format to specify the foot-plant tags. For example, *<footplant> <5, 10, 15> </footplant>* specifies that joint number 5 must be planted on the ground between frame numbers 10 and 15. Since we use the same cyclic motion repeatedly, the tags from the original motion need to be recalculated and added automatically. Then, given a list of tags, the IK solver computes the knee and ankle angles during the frames that require foot planting.

If we change only a particular point on the space-time warp curve, we have an unnatural speed change in the middle. We achieve consistency by applying Gaussian filtering to the space-time warp curve, such that neighboring points are also influenced by the changes. This process is explained in Fig. 6.



Construct space-time warp curve
User's changes on the space-time warp curve
Apply Gaussian filtering
Root joint position adjustment
Apply IK solver

Fig. 6. Speed changes using space-time warp curve.

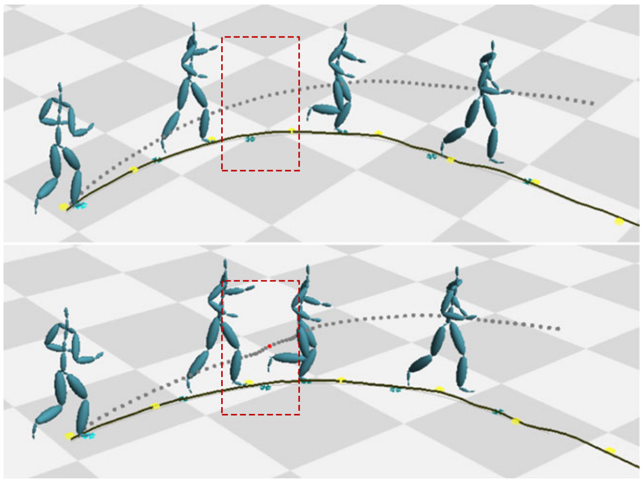


Fig. 7. Example of changes on space-time warp curve. Red point indicates target point.

Figure 7 shows an example in which a space-time warp curve is applied to change the motion speed. Note that the red dots indicate the target points of the space-time warp curve. After repositioning the target points, the entire curve is recalculated to reflect the speed changes.

If we allow complete freedom for changes on the space-time warp curve, it may produce unnatural motion, because the IK solver does not guarantee quality motion if the new speed differs too much from the original. We guarantee good quality motion by placing a constraint on the motion speed so that the speed can change between a minimum of −50% and a maximum of 200% through the space-time warp curve.

4. Collision Detection Using the Space-Time Warp Curve

When working with crowd-motion synthesis, the most annoying problem is checking whether the synthesized motions generate any collisions and where the collisions occur. Since our goal is to achieve realistic low-level individual motion as well as high-level controllable crowd formation, we need to detect a collision as fast as possible by either changing the timing or trajectory of the motions to avoid any collisions.

In our approach, the spatial trajectories are determined through the hub-sub-control-point scheme. Therefore, changing the control points to avoid collisions at a particular point may not be appropriate because changing the trajectory shape may cause other collisions in other places. Instead, our aim is to avoid a collision by changing the temporal motion speed. In this regard, our space-time warp curve has an advantage: it can visualize the collision because the *x*- and *z*-coordinates represent the 2D position on the ground and the *y*-coordinate is the temporal location.

Figure 8 shows two space-time warp curves. Even though the two cubic B-spline curves intersect in the middle, their space-time warp curves do not intersect, which means that the two motions do not result in a collision.

Since each character has a volume, rather than being a point, two characters' body parts may collide with each other even though their space-time warp curves do not intersect. We solved this problem by setting a minimum-distance threshold value to test for intersection. Therefore, if the distance between two points from two separate space-time warp curves is within the threshold value, we regard it as an intersection.

When we create a large number of motions for crowds through a hub-sub-control-point-based spline curve, it is impossible to adjust the motion speed for every intersection point to avoid collisions if we must manually change the shape of the space-time warp curve. Our algorithm automates this process by finding the intersection and changing the speed incrementally until no intersections remain. When we detect an intersection point

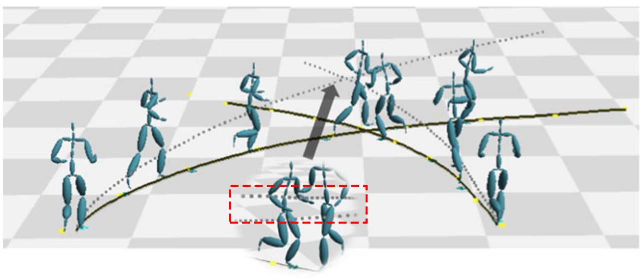


Fig. 8. Collision test by checking intersection between two space-time warp curves. Red box shows scene zoom-in.

between two space-time warp curves, we choose one of the curves and change the speed immediately before the intersection. Figure 9 shows an example of collision avoidance. More details on the collision avoidance can be found in the following pseudo-code:

```
int N // Number of curves
Curves C[N]; // Array containing curves
int Ti, Tj;
bool collision;
While (!collision)
  Collision = false;
  For i = 0 to N
    For j = 0 to N
      If (IntersectTest(C[i],C[j]))
        collision = true;
        Ti = i; Tj = j;
        Break;
      End If
    End For
  End For
  If (!collision) break;
  Else AdjustSpeed(Ti,Tj);
End While
```

IV. Experiments

We validated our algorithm through a series of experiments. The computing environment included an Intel i7 CPU PC with 8 GB RAM and an Nvidia Geforce GTX 560 graphics card. Characters were rendered using the OpenGL library and the Fast Light ToolKit library for user-interface design. The input data was a cyclic running motion consisting of 27 frames. The motion-data format was BioVision's BioVision Hierarchy [14]. The resulting video was uploaded on YouTube at https://youtu.be/aV9TY3rXSvw.

The first experiment tested the ability of our hub-sub-control-point scheme to control a cubic B-spline curve. Figure 10 shows the result of our test. In this experiment, the initial scribble was almost a straight line, but after we fit the scribble with the cubic B-spline, we modified the curve by changing the hub-control point. We preserved the offset distances between the hub-control point and the sub-control point even after we changed the position of the hub-control points, which means that the positions of the sub-control points were set and changed automatically, based on their pre-defined offset distance to the hub-control points.

Furthermore, for the first half of the curve, the offset distance between the sub-control points and the hub-control points was set to 2.0 and increased to 5.0 for the second half of the curve. This results in a dispersion of the curves toward the end. That is, the minimum-distance constraint between the hub-control point and sub-control points was set to 2.0.

The second experiment was designed to evaluate the natural motion-synthesis algorithm. We compared two cases. The first case involved using only the tangent angle of the curve for root-joint orientation without a disparity map. The second case used the tangent angle of the curve along with the disparity-map technique proposed in Section II.

The result showed that the disparity map produced more natural motions, since we accounted for the difference between the current and original motion by considering the curve angle. Thus, the disparity map produced more-natural motions with enhanced stability. The comparisons can be found on the YouTube video clip. Figure 11 shows a screen shot.

The third experiment tested the group-motion synthesis. We drew two scribbles and fit them with the hub-sub-control-point scheme, which automatically generated three additional curves. Each curve was then used to synthesize the motion. In this process, the algorithm automatically checks the intersection between the space-time warp curves and reduces the speed until no further intersections exist. Figure 12 shows the result.

For the final example, we set 30 characters in motion using our technique. Figure 13 shows a screen shot of the

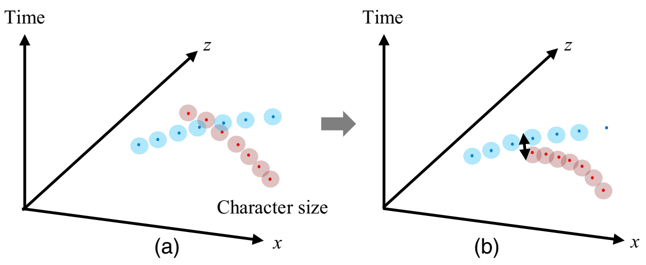
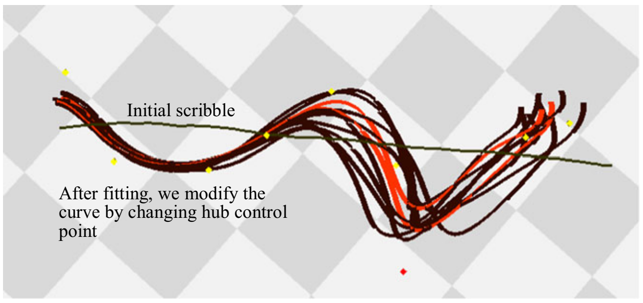


Fig. 9. (a) Before: two time-warp curves intersect in middle. (b) After: second (red) time warp changes its shape to reduce speed and avoid intersection.


Initial scribble
After fitting, we modify the curve by changing hub control point
Fig. 10. Trajectory creation through spline curves based on hub-sub-control-point scheme.

movement. A more detailed animation can be found in the YouTube video. In this scenario, all the characters attempted to maintain a constant speed for as long as possible. However, they changed their speed automatically to avoid collisions. In complicated situations, some collisions could not be prevented automatically. In those cases, we allowed the user to change the speed manually to avoid a collision.

Table 1 presents the performance of our algorithm as a function of the number of characters. The motion-synthesis times vary depending on the complexity of the scene. We established that the most time-consuming step involved changing the speed to avoid a collision, since the algorithm must evaluate many different speed adjustments until no intersection remains between any two space-time warp curves (Table 2).

One limitation of our algorithm is that we only apply our technique to locomotion data. In the future, we hope to extend our algorithm to more general types of motion, for example, those associated with dancing or sports. Additionally, through a small upgrade, we would like to extend our algorithm to support characters of different sizes.

We currently avoid collisions using a simple, inefficient brute-force algorithm. As a result, the character-formation constraint cannot be satisfied because the character speed is automatically adjusted on the fly to avoid collisions, which makes it hard to maintain the formation. If we want to set the formation constraints, we can cast the problem as an optimization and solve it using a mathematical optimization technique. We plan to design a more sophisticated collision-avoidance algorithm in the future.

Another limitation of our technique is that it only works on a flat surface because of the following reasons. First, we used an input cyclic motion captured only on a flat surface. Second, we used the y-axis as the time domain. If we embed space-time warp curves in an uneven environment, the simulation does not work properly because the characters may not stay in the same time domain, depending on their location on the uneven terrain.

To solve the first problem, we believe that we can extend our algorithm by combining it with other techniques; for example, motion-graph techniques or parameterized motion-blending techniques, which use different styles of input motions. To address the second problem, we can use a separate imaginary time space where all characters can exist at the same time. In that case, however, we must sacrifice our advantage of embedding space-time warp curves directly into the spatial environment.

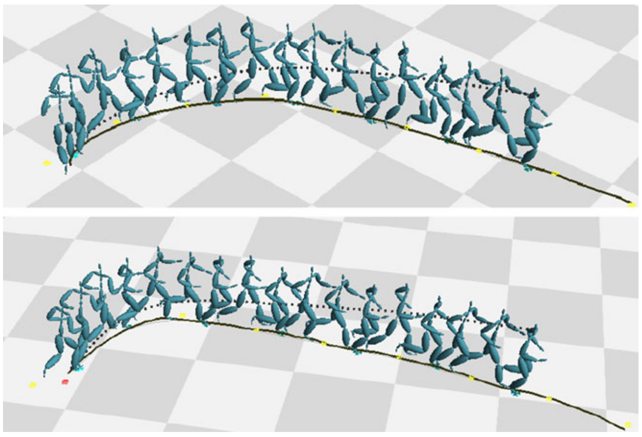


Fig. 11. Two-level motion synthesis using disparity map.

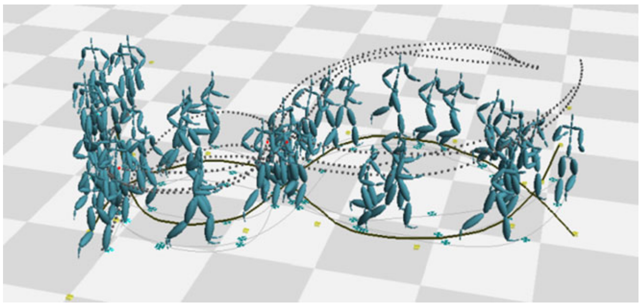


Fig. 12. Example of group-motion synthesis.

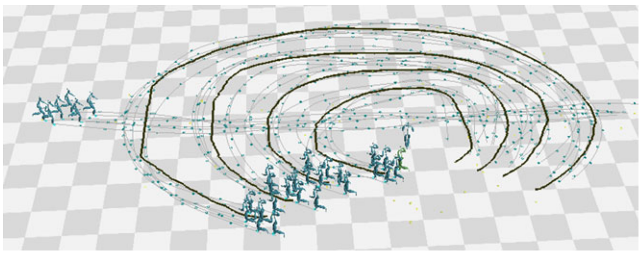


Fig. 13. Example of motion synthesis with 30 characters.

Table 1. Our algorithm's performance.

# of characters	Motion-synthesis time	Avg. # of frames
1	1 s	120
10	10 s	250
30	40 s	450
50	3 min 12 s	800

Table 2. Comparison between manual trajectory specification and the hub-control-point scheme.

Amount of times	Manual specification	Hub-control-point scheme
# of trajectories	25.4 s	1.2 s

V. Conclusion

In this paper, we introduced a new motion-synthesis algorithm for multiple characters. At the high level, our algorithm introduced scribble-based initial paths, which were transformed into spline curves using the cubic B-spline curve-fitting technique. We created a set of different trajectories by introducing the hub-sub-control-point scheme. Then, our algorithm generated a space-time warp curve for each path to change the speed motion automatically. The space-time warp curve offers advantages in terms of visualizing collision detection as well as providing a more user-friendly interface.

In Section IV, we compared the total amount of time when we specified the trajectory manually with the time when we used our hub-control scheme. It turns out that our method created the trajectories 10 times faster than those created manually.

References

[1] C.W. Reynold, "Steering Behaviors for Autonomous Character," in *Proc. Game Developers Conf.*, San Jose, CA, USA, 1999, pp. 763–782.

[2] D. Helbing and P. Molnar, "Social Force Model for Pedestrian Dynamics," *Phys. Rev. E*, vol. 51, Jan. 1995.

[3] A. Treuille, S. Cooper, and Z. Popovic, "Continuum Crowds," *ACM Trans. Graph*, vol. 25, no. 3, July 2006, pp. 1160–1168.

[4] T. Kwon et al., "Group Motion Editing," *ACM Trans. Graph*, vol. 27, no. 3, Aug. 2008, pp. 80:1–80:8.

[5] M. Kim et al., "Synchronized Multi-character Motion Editing," *ACM Trans. Graph.*, vol. 28. no. 3, Aug. 2009, pp. 79:1–79:9.

[6] J. Kim et al, "Interactive Manipulation of Large-Scale Crowd Animation," *ACM Trans. Graph.*, vol. 33, no. 4, July 2014, pp. 83:1–83:10.

[7] H.P.H. Shum, T. Komura, and S. Yamazaki, "Simulating Multiple Character Interactions with Collaborative and Adversarial Goals," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 5, May 2012, pp. 741–752.

[8] A. Barnett, H.P.H. Shum, and T. Komura, "Coordinated Crowd Simulation with Topological Scene Analysis," *J. Comput. Graph. Forum*, vol. 35, no. 6, Oct. 2016, pp. 120–132.

[9] B. Yersin et al, "Crowd Patches: Populating Large-Scale Virtual Environment for Real-Time Applications," *Proc. Symp. Interactive 3D Graph. Games*, Boston, MA, USA, Feb. 27–Mar. 1, 2009, pp. 207–214.

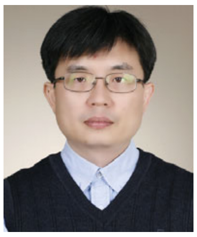
[10] P. Shirley, M. Ashikhmin, and S. marschner, *Fundamentals of Computer Graphics*, 3rd Edition, Boca Raton, FL, USA: CRC Press, 2009.

[11] S. Leon, *Linear Algebra with Applications*, 3rd Edition, London, UK: MacMillan Publishers, 1990.

[12] M. Gleicher, "Motion Path Editing," *Proc. Symp. Interactive 3D Graph.*, Triangle Pk, NC, USA, Mar. 19–21, 2001, pp. 195–202.

[13] L. Kovar, J. Schreiner, and M. Gleicher, "FootSkating Cleanup for Motion Capture Editing," *Proc. ACM SIGGGRAPH/Eurograph. Symp. Comput. Animation*, San Antonio, TX, USA, July 21–22, 2002, pp. 97–104.

[14] BioVision Hierarchy (BVH) Format, Accessed 2016. http://www.character-studio.net/bvh_file_specification.htm



Mankyu Sung received his BS degree in computer science from Chungnam National University, Daejeon, Rep. of Korea, in 1993, and his MS and PhD degrees in computer science from the University of Wisconsin-Madison, WI, USA, in 2005. From January 1995 to July 2012, he worked for the Digital Contents Division of the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. He has been an assistant professor with the Dept. of Game and Mobile Contents, Keimyung University, Daegu, Rep. of Korea, since March 2012. His current research interests include computer graphics, computer animation, computer games, and human-computer interaction. He is a member of the ACM.

Gyu Sang Choi received his PhD degree in computer science and engineering from Pennsylvania State University. He was a research staff member at the Samsung Advanced Institute of Technology in Samsung Electronics, Suwon, Rep. of Korea, from 2006 to 2009. Since 2009, he has been with Yeungnam University, Gyeongsan, Rep. of Korea, where he is currently an assistant professor. His research interests include embedded systems, storage systems, parallel and distributed computing, supercomputing, cluster-based Web servers, and data centers. He is now working on embedded systems and storage systems, while his prior research has been mainly focused on improving the performance of clusters. He is a member of the IEEE and ACM.