SPECIAL ISSUE PAPER

Characteristics preserving racer animation: a data-driven race path synthesis in formation space

Charlie Irawan Tan and Wen Kai Tai*
National Dong Hwa University, Shoufeng, Taiwan

ABSTRACT

We propose a race path synthesis framework based on a data-driven approach that provides good controllability for synthesizing race paths with characteristics preserved for racer animations. We introduce formation field, a data structure that samples regions in formation space that contains formations of exciting and realistic race paths, generated using a set of collected race paths in a path database. By traversing the regions according to a given constraint, we generate a path in formation space that defines how to synthesize the desired race path by interpolation. Because the new race path is synthesized from existing paths with quality guaranteed, it also provides the same level of quality. As the experimental and user study results show, our framework produces good results effectively and is suitable for both real-time applications such as horse racing games and race-path-generating tools. Copyright © 2012 John Wiley & Sons, Ltd.

KEYWORDS
race path synthesis; formation field; racing game

*Correspondence
Wen Kai Tai, National Dong Hwa University, Sec. 2, Da Hsueh Rd., Shou-Feng, Hualien, 974, Taiwan.
E-mail: wktai@mail.ndhu.edu.tw

1. INTRODUCTION

Racing game, one of the oldest genre in the history of video games that still exists and evolves until now, promises many research topics to be discovered. In some of the racing games [1,2], the race path is already decided in advance by another module. The generated race path must demonstrate a realistic and exciting race animation so that the players will enjoy a fascinating gameplay experience. To have a believable race for players, the game must be able to exhibit a sequence of user-expected events. For instance, if a player uses a back-finisher strategy (a horse runs slow early and quickens in the latter part of the race), the game should create a race where the player’s horse exhibits back-finisher behavior by deciding the ranking of the horses at some specific times in the race. In this paper, we propose a framework to utilize a race path database and synthesize realistic race paths with characteristics preserved for racer animations that satisfy user constraints.

Our framework, as shown in Figure 1, utilizes a data-driven approach to synthesize race paths from a race path database. The main idea of the data-driven race path synthesis is to reuse existing race path samples to form a new plausible race path. One of the advantages of our method is the capability to preserve the characteristics of sample race paths in the synthesized race paths. Because the high-quality samples of race paths can be selected offline and then used later in the real-time synthesis process, this method is capable of synthesizing high-quality race paths in a limited amount of time. By replacing the sample race path database (SRPD), we can easily synthesize race paths with specific characteristics from any types of racing. Data-driven race path synthesis also provides controllability with low computation time as shown in our results. It can synthesize a race path that satisfies a specific ranking constraint in real time, which is hard to achieve by conventional methods such as simulation. Unfortunately, the robustness of the data-driven race path synthesis is limited by the quality and the quantity of SRPD. Robustness of the race path synthesis method relies on its capability to satisfy user-specified constraints, that is, to increase robustness, a larger SRPD is necessary. In this paper, we propose a novel data-driven race path synthesis method that provides good robustness by utilizing interpolation of the sample race path data.

To our knowledge, there is no description that can precisely define nor a metric that can estimate the quality of a race path. We quantified the quality of our synthesized race paths through a user study. From our observations and comments from participants of the user study, some characteristics that define a good-quality race path are frequency of change in rankings, especially near the
Race path synthesis in formation space  

C. I. Tan and W. K. Tai

finish line; turnover event, that is, the racer who ranked last takes over the first place at the last time; and the difference in the finish time between the champion and the runner-up. In other words, the quality of a race path is closely related to its racers’ ranking changes. This inspires us to design a race path synthesis method that can easily control the ranking changes in a race by specifying the positions of the racers at certain times in the race to indirectly control the quality.

The remainder of this paper is organized as follows. In the next section, we discuss related work. We introduce our data structure for race path synthesis in the formation field section. In the race path synthesis section, we describe our synthesis method in detail and show the result in the results section. Finally, we discuss limitations, applications, and future work in the discussion and future work section.

2. RELATED WORK

Several researchers surveyed theoretical and practical results in path finding [3–5]. Togelius and Lucas compared controllers used in miniature racing car simulations [6] and investigated the trade-off between optimizing controller performance on a particular race track and increasing its robustness to handle new tracks [7]. A simulation of human control of a motocross racing game using a neural network is conducted by Chaperot and Fyfe [8]. All the aforementioned racing-related path researches were performed with the objective of optimizing the finishing time of a single racer. In a work more closely related to ours, Tan et al. proposed a framework to simulate race paths for several racers [9,10] using sequential execution of the A* algorithm [11]. Although their method is fast enough to be implemented in a racing game, their work lacks controllability of ranking in the path.

The research on crowd animation was initially carried out to generate believable crowd movement [12,13]. A graph representing the clearance and proximity information for each agent is used by Sud et al. [14]. The term formation is introduced by Kwon et al. [15], which represents the positions of each agent and is used to provide good control over the crowd movement. A recent study related to our work was conducted by Takahashi et al. [16], who proposed a method to generate crowd locomotion by transforming a given formation into another formation. Each formation and its adjacency relation are represented by a mesh, and by applying a mesh deformation technique in the frequency domain, they synthesized a smooth locomotion from one formation to another. Their approach cannot be directly applied to synthesize a race path. The first reason is that, in a race, the adjacency relationship between racers is not necessarily preserved as they are in the research of Takahashi et al., and preserving the adjacency might degenerate the quality of the produced path. Second, we cannot guarantee the quality of the interpolation between formations because they are generated at real time. Finally, their method works best for interpolating two similar to moderately different formations; more constraints are required when users want to synthesize a good-quality race path with transitions between two completely different formations. On the other hand, our framework requires fewer constraints (as few as one constraint at the finish or none when no specific ranking is required) without reducing the quality of the synthesized race path.

A data-driven race path synthesis method proposed by Tan and Tai [17] divides sample race paths into smaller clips and rearranges their sequence to form a new race path. Although their work is capable of satisfying user-specified constraints, their computation time is high because of the lack of a supporting data structure. Furthermore, with the synthesis process limited to rearranging clips of sample race paths, their method does not fully utilize the reusability of the sample, and hence its robustness is reduced. One possible data structure that is suitable for their method is a graph structure, similar to the motion graph that is used in motion synthesis [18–20]. The disadvantage of a graph structure is that it can only synthesize race paths when the constraints are corresponding to clips’ first or last formation. We propose a data structure that can synthesize any kind of constraints as long as it corresponds to a formation that is contained in SRPD.

3. FORMATION FIELD

We extend the definition of racetrack and race path in the work of Tan and Tai [17], summarized in Table I and
Table I. Definitions.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>( \mathcal{P} = {P_0, \ldots, P_m} )</td>
<td>A set of sample race path data</td>
</tr>
<tr>
<td>Race path</td>
<td>( P_t = (f'<em>0, \ldots, f'</em>{l-1}) )</td>
<td>A sequence of formations sampled per unit of time</td>
</tr>
<tr>
<td>Formation</td>
<td>( f = (R, S, \beta, \theta, L(f)) )</td>
<td>A sextuple: racers’ deployment, speed, barycenter, direction, and distance to the finish</td>
</tr>
<tr>
<td>Deployment</td>
<td>( R = (r_0, \ldots, r_{n-1}) )</td>
<td>A sequence of points, each representing a racer’s position in the formation’s local coordinate system, centered at the formation’s barycenter</td>
</tr>
<tr>
<td>Ranking</td>
<td>( rk(R) = (id_1, \ldots, id_n) )</td>
<td>A sequence of racer’s id, where ( id_1 ) is the foremost racer’s id and ( id_n ) is the last</td>
</tr>
<tr>
<td>Speed</td>
<td>( S = (s_0, \ldots, s_{n-1}) )</td>
<td>A sequence of racer’s instantaneous speed</td>
</tr>
<tr>
<td>Barycenter</td>
<td>( \beta )</td>
<td>The average of racers’ position, barycenter</td>
</tr>
<tr>
<td>Formation’s direction</td>
<td>( \theta )</td>
<td>Direction of the x-axis of a formation’s local coordinate with respect to the global x-axis</td>
</tr>
<tr>
<td>Reference point</td>
<td>( q )</td>
<td>The point on the racetrack’s reference curve, nearest to the formation’s barycenter, ( \beta )</td>
</tr>
<tr>
<td>Finish point</td>
<td>( q_{\text{fin}} )</td>
<td>The point on the racetrack’s reference curve intersecting with the finish line</td>
</tr>
<tr>
<td>Distance to the finish</td>
<td>( L(f) )</td>
<td>The arc length of the reference curve from ( q ) to the finish point, ( q_{\text{fin}} )</td>
</tr>
</tbody>
</table>

where \( r^i_k \) and \( r^j_k \) denote the position of the \( k \)th racer of deployments \( R_i \) and \( R_j \), respectively, and \( d \left( r^i_k, r^j_k \right) \) denotes their Euclidean distance. Note that the index \( k \) of each racer is fixed on the basis of its position at the starting line (we also refer to index \( k \) as the racer’s index). We only record a single instance of duplicated deployments, that is, for any \( R_i, R_j \) is removed if \( D(R_i, R_j) < \epsilon, \epsilon \approx 0 \), and \( R_j \) is recorded as a neighbor of \( R_i \) if \( D(R_i, R_j) < \infty \).

Collision detection in an interpolation between two deployments can only be performed after the deployments are converted to formations, that is, after the position of their barycenters in the racetrack is decided in the synthesized race path according to user constraints. Unfortunately, user constraints are given at real time and unavailable when we generate the formation field. We approximately detect the collision in the \( xz \) time space and use a parallelepiped to represent interpolations of a racer in the deployments. Colliding parallelepipeds mean that collision between racers will happen in the interpolation of the deployments. The bottom and top faces of a parallelepiped

Figure 2. An illustration of terms in Table I.

Figure 3. Deployment interpolation’s collision detection, with (a) and without collision (b).
are rectangles centered at racer positions \( r_i^j \) and \( r_i^{j+1} \), respectively. Reasonably, the rectangles align to the racer’s front vector \( \frac{r_i^j - r_i^{j+1}}{\| r_i^j - r_i^{j+1} \|} \), and we restrict the angle between the front vector and the formation’s local \( x \) positive axis to \( \alpha \), that is, we constrain the front vector to \( (\cos(\alpha), \sin(\alpha)) \) if the angle is greater than \( \alpha \), where \( \alpha \) is the largest turning angle of a racer within two consecutive formations defined in a race physics parameter. Figure 3 illustrates an interpolation of two formations with three racers that will and will not cause collision.

### 4. RACE PATH SYNTHESIS

Race path synthesis proceeds in real-time stages and consists of three steps: constraints specification, candidate deployment selection, and postprocessing. Given the constraints, we select qualified deployments and use them for synthesizing a race path. We calculate the number of formations in the synthesized path and synthesize them at the postprocessing.

#### 4.1. User Constraints Specification

Constraints are specified as a sequence of key formations \( C_t \) or key rankings \( C_t \). An element \( c_i \in C_t = (c_0, \ldots, c_m) \) contains a formation \( f_i \) located \( l_i \) meters away from the finish line, \( L(f_i) = l_i \). Users can use key formations to precisely specify any desired formations to synthesize a race path for applications that require high precision such as race paths in a movie. Unfortunately, it often leads to a labor-intensive work, and automatic specification of each racer’s position for a racing game is challenging to implement.

Alternatively, users can use a key ranking sequence \( C_t = (c_0, \ldots, c_m) \) as constraints. Each \( c_i \) consists of an n-tuple \((i_1, \ldots, i_n)\) and the desired distance \( l_i \) to the finish line, where \( i_1 \) denotes the racer’s index and \( k \) represents the rank. Defining a ranking sequence as constraints is more feasible for users and facilitates the implementation of a racing game.

The distance between two consecutive user-specified constraints can be calculated as \( d_{i,j} = L(f_i) - L(f_j) \) for \( c_i, c_j \in C_t \) or \( d_{i,j} = l_i - l_j \) for \( c_i, c_j \in C_t \). We search the corresponding key deployment sequence \( \mathcal{R} = (R_0, \ldots, R_m) \) for user-specified constraints in formation field and find candidate deployments as described in the next subsection. The corresponding key deployment for \( c_i \in C_t \) is a deployment \( R_i \) that satisfies \( r_k(R_i) = c_i = (i_1, \ldots, i_n) \). Accordingly, the corresponding key deployment for \( c_i \in C_t \) is a deployment \( R_j \) such that \( D(R_i, R_j) \approx 0 \), \( R_i \in f_i \in c_i \).

#### 4.2. Candidate Deployment Selection

We formulate a candidate deployment selection problem as a path-finding problem in formation space. To satisfy the constraints, the distance of the path connecting two consecutive key deployments \( R_i \) and \( R_j \) where \( R_i, R_j \in \mathcal{R} \) must be less than or equal to the user-specified distance \( d_{i,j} \) for two consecutive constraints \( c_i \) and \( c_j \). \( \mathcal{R} \) is treated as checkpoints, and our purpose is to find the shortest path that passes the checkpoints sequentially. The resulting sequence of deployments on the path is the candidate deployments.

The A* algorithm [11] is used to find the shortest path from two consecutive key deployments \( R_i \) to \( R_j \) where \( R_i, R_j \in \mathcal{R} \). Initially, \( R_i \) is the initial node, and the open set of A* algorithm is filled with its neighbors as recorded by the formation field. As the node is changed to the deployment with minimum estimated total cost in the open set, we add neighbors of the deployment to the open set. Equation (1) is used as the heuristic estimate.

Before we proceed with the process to select the actual deployments that will be used to interpolate synthesized deployments, we introduce the concept of interpolation primitive first. The interpolation primitive of d-dimensional formation space is defined as \( \mathcal{I} = \bigcup_{k=1}^{d} I_k \), where \( I_k \) is a simplex [21] of k-dimensional space. Each simplex represents the approximated region that contains formations of an exciting race path, whereas its vertex represents a deployment in formation space sampled in SRPD. Using a high-dimensional simplex as the interpolation primitive will provide more variety of interpolation results at the cost of higher computation time. Let \( \mathcal{I}_i,j \) be the sequence of interpolation primitives that encloses the shortest path from key deployments \( R_i \) to \( R_j \) \( (R_i, R_j \in \mathcal{R}) \). We define the length of an interpolation primitive as the length of the shortest path, and the deployment interpolation is performed along the shortest path. An interpolated deployment corresponding to a point in the shortest path can be represented by a barycentric coordinate defined by the enclosing simplex’s vertices of the shortest path. Using the point’s barycentric coordinate as weight, we interpolate the deployment from the simplex’s vertices. Figure 4 illustrates the deployment interpolation process, and Figure 5 illustrates primitives in formation space and the corresponding deployments in two-dimensional Euclidean space, where \( r_i^k \) and \( r_j^k \) denote racers in \( R_i \) and \( R_j \), respectively, whereas \( k \) denotes the racer’s index.

![Figure 4. Deployment interpolation using two-dimensional interpolation primitives. \((w_i, w_j, w_k)\) denotes the barycentric coordinate of the interpolated deployment (red point). The shortest path (blue line) will be divided into \( \mu \) points in the postprocessing step.](image-url)
C. I. Tan and W. K. Tai  
Race path synthesis in formation space

Figure 5. The blue line shows the shortest path in formation space where deployment interpolation is carried out. The red-dotted line shows the shortest path if a one-dimension primitive is used instead of a two-dimension or three-dimension primitive, which is longer than the blue line. The bottom row shows the area of possible racers’ position that can be interpolated from the respective interpolation primitives. More variety of interpolation results can be generated by higher-dimensional interpolation primitives because they cover a larger region in formation space.

<table>
<thead>
<tr>
<th>Interpolation primitive in formation space</th>
<th>1D</th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Racers’ position in two-dimensional Euclidean space (e.g. on a racetrack)</td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
</tbody>
</table>

4.3. Postprocessing

In the previous step, we generate a sequence of interpolation primitive $I_{i,j}$ for two consecutive key constraints $c_i$ and $c_j$. The number of deployments that should be generated for $I_{i,j}$ is calculated as $\mu_{i,j} = d_{i,j}/\bar{v}$, where $\bar{v}$ is the average speed for a specific race type and $d_{i,j}$ is the user-specified distance between the constraints. We then interpolate the deployment using each interpolation primitive sequence $I_{i,j}$ by first dividing its shortest path (Figure 4) to $n_{i,j}$ points. Using each point’s barycentric coordinate as weight, we performed the deployment interpolation by using the sample deployments defining the simplex enclosing the point.

We calculate the density of an interpolation primitive sequence $I_{i,j}$ as $\rho_{i,j} = \mu_{i,j}/\lambda(I_{i,j})$, where $\lambda(I_i)$ is the length of the shortest path connecting key constraints $I_i$.

Figure 6. Interpolation primitive replacement. The blue lines represent the shortest path, and the red lines represent $I_a$ and $I_b$, whereas red points are $N_i$. We ignore $R_k$ in (a) because it does not lead to a shorter path (note that $D(R_i, R_j) = \infty$). In contrast, $R_k$ in (b) leads to a shorter path.
applied to synthesized race path. In contrast, with low entropy and hence might reduce the excitement rate of the synthesized race path. We warn users if $\rho_{i,j}$ goes below a threshold $\rho_{\text{min}}$, noting that distance between the constraints is too close to have a good interpolation result. On the other hand, if $\rho_{i,j}$ goes above a threshold $\rho_{\text{max}}$, we insert other interpolation primitives until their total density goes in between $\rho_{\text{min}}$ and $\rho_{\text{max}}$, that is,

$$\rho_{\text{min}} < \frac{\mu_{i,j}}{\lambda (I_{i,j})} + \sum_{k} m \lambda (I_{k}) < \rho_{\text{max}}$$  \hspace{1cm} (2)

where $I_{k}$ is the new interpolation primitive and $m$ is the number of new interpolation primitives. We only add interpolation primitives located around $I_{i,j}$ in the formation space, that is, we use sample deployments around $R_{i}$ and $R_{j}$ to generate the new interpolation primitives.

After interpolation, we translate and rotate the sequence of synthesized deployments to a user-defined racetrack using the method set out by Tan and Tai [17].

### 4.4. Implementation Detail

To facilitate key ranking constraints, we generate a ranking table that maps a ranking to a list of deployments. For a quick deployment query in the key formation constraint, we generate a hash table. An all-pairs shortest-path table can be used instead of an A* algorithm in real time as described in the candidate deployment selection subsection to achieve an instant response of path synthesis. Unfortunately, in most cases, the table cannot be loaded to the main memory.

Although using a higher dimension of interpolation primitive provides more variety of synthesized formations, our application requires real-time processing. Therefore, we use an interpolation primitive only up to three dimensions (a tetrahedron) and use a greedy approach when optimizing the interpolation primitive combination.

To facilitate user interactions to the synthesis method, formation space is traversed on the basis of user inputs, instead of using path finding from a starting deployment to a target deployment. Let $R_{s}$ be the current deployment before user interaction. User interaction to the $i$th racer is applied to $R_{s}$ by translating $r_{i}^{s} \in R_{s}$ to generate $R_{i}^{'}$. We calculate $D \left( R_{i}^{s}, R_{i}^{'} \right)$, where $R_{i}$ is the neighbors of $R_{s}$, and select the neighbor with the smallest distance, $R_{i}$. We then interpolate the next deployment $R_{i+1}^{s}$ by interpolating $R_{i}$ to $R_{i}^{'}$. We can mimic a game artificial intelligence (AI) by further modifying $R_{i}^{'}$ and constraining it so that $r_{i}^{s'}$ has a specific ranking.

### 5. RESULTS

Our experiments were performed on a personal computer with a Pentium Core i7 2.8-GHz central processing unit and 6-GB main memory. Our implementation of formation field generation is a single-threaded Win32 console program. We use CGAL library for collision detection before calculating Equation (1). SRPD containing 3000 race paths is generated using the method proposed by Tan et al. [9,10]. Each race path is about 2000 m long and consists of about 149 formations with eight racers, selected manually. We extracted a formation field with 335 217 sample deployments. Table II shows memory consumption of the formation field. The generation process requires roughly 15 hours, dominated by similarity calculation because of the large number of formations in SRPD.

We implemented our real-time race path synthesis method in a horse racing game. Our method synthesizes a race path on the basis of the given key rankings by game logic at start, some on the middle parts, and at finish. Figure 7 shows some screenshots of the game. The game also features a race path generator that we used to demonstrate the controllability features of our method. We specified spatial constraints using six novel key formations (e.g., diamond, triangle, etc.) at specific locations. As shown in Figure 7 (top left), the racers managed to form the specified formation at the location (marked by blue rectangles). Although user interaction is not the purpose of using the data-driven race path synthesis method, our method is capable of generating a race path with user interference as seen in a racing game where game players control a racer and interact with the AI in the game. We can utilize the formation field to mimic a game AI by stochastically allowing or restricting player’s movements as described in the implementation detail subsection. We managed to add user interference to the game where a player can control one of the horses as shown in Figure 7 (bottom). Please refer to the accompanying videos for our experiment on synthesizing a race path using key formations/rankings, user interaction, and AI simulation (https://sites.google.com/site/formationfield/).

The game is also used to conduct a user study as the reference regarding the quality of the synthesized race paths. Using the same hardware specification as the formation field generation process and a 24-in. monitor, we showed our test path set to 20 participants with different educational backgrounds and genders. Thirty race paths in the test set consist of paths synthesized from our framework, paths from the database, digitalized paths from real race data [22], and a set of control data, each with about 1-minute duration. The control data contain some good-quality race paths (manually designed by the author), and some race data show races with no change of ranking at

<table>
<thead>
<tr>
<th>Structure</th>
<th>Memory (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formation field</td>
<td>168,173.60</td>
</tr>
<tr>
<td>Ranking table</td>
<td>195.66</td>
</tr>
<tr>
<td>Hash table</td>
<td>269.03</td>
</tr>
</tbody>
</table>

Table II. Memory requirement for each structure used by the formation field.
Figure 7. Screenshots from our horse racing game. The top-left image shows that our framework can satisfy any formation constraint, in this case, an “X.” The bottom-left image shows a formation when the game is set to allow any player movement; the bottom-right image is a formation when the game is set to interfere with player movement (the red arrow illustrates user’s input).

Table III. The result of a user study on our synthesized race paths.

<table>
<thead>
<tr>
<th>Path type</th>
<th>Average</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesized</td>
<td>7.42</td>
<td>0.63</td>
</tr>
<tr>
<td>Database</td>
<td>8.37</td>
<td>0.59</td>
</tr>
<tr>
<td>Real</td>
<td>8.27</td>
<td>0.52</td>
</tr>
<tr>
<td>Control high</td>
<td>8.23</td>
<td>0.46</td>
</tr>
<tr>
<td>Control low</td>
<td>1.62</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Control high and low are the score of control data.

at the finish, a motion picture showing fierce competition requires more constraints.

6. DISCUSSION AND FUTURE WORK

We proposed a data-driven race path synthesis approach that provides good controllability and is effortless to use. A collection of good-quality paths available from sources including videos, simulations, and so on is used to construct the formation field so that the synthesized paths are guaranteed to have quality as good as the race paths in SRPD. Because our method is benefited from a wide variety of formations in SPRD, a sample race path with intense ranking change is preferred. We also recommend a sample race path whose racers apply different kinds of strategy. A user study was conducted to verify the quality of our framework. As shown in the experimental results, our framework performed well enough to be applied in real-time applications such as horse racing games.

In some uncommon cases, the formation field might be unable to satisfy user constraints because of insufficient data in the database or because users specify invalid constraints because of a low value of $d_{i,j}$ (for example, please refer to Figure 9). We warn users when such cases happen. If the desired formation is unavailable in SRPD and high computation time is allowed, we apply the formation interpolation method [16] to add a formation to the formation field.

Formation similarity calculation is the most time-consuming process when generating a formation field. Reducing the required time for this process by utilizing...

all after the racers depart 10 m from the start line. We showed the race paths in random order and let the participants rest for 5 minutes after a session of 15 minutes to reduce learning effects and fatigue. The participants were asked to score the race paths with ratings ranging from 1 to 10. As shown in Table III, the synthesized paths obtain satisfactory results. The score for the control data shows that although there is no exact method to quantify race path quality, most participants had similar preference regarding the quality of a race path. An interesting factor mentioned by some participants is that they were hardly excited by the race because they did not get involved in the race because, for example, none of the virtual horses involved in the race were owned by them.

Increasing the number of constraints increases the computation time slightly, as shown Figure 8. The experiment is conducted by synthesizing 50 race paths (1400 m) for each number of constraints for both key formation and key ranking constraints. The number of constraints needed to synthesize a new race depends on the application. Although a racing game requires only one constraint...
Race path synthesis in formation space
C. I. Tan and W. K. Tai

parallel computing paradigm such as general-purpose computing on graphics processing units is one of our future work. We are also interested in investigating the factors that affect the quality of a race path. We would develop an automatic system that can select race paths with good quality to develop SRPD by understanding those factors.

Several applications are possible for our framework. In virtual life games, for example, horse racing games, our method can be implemented for its racing simulation. For motion picture productions, it can be used to depict characters participating in racing where certain rankings are required. Our framework is applicable as well in other applications where control over the crowd is important and necessary (assuming that the crowd is moving in a simple environment).

ACKNOWLEDGEMENTS

The authors would like to thank all participants who contributed to this study. Also, we are grateful to anonymous reviewers for their helpful comments. This work was partially funded by the National Science Council, Taiwan, ROC, under grant NSC99-2221-E-259-021.

REFERENCES


AUTHORS’ BIOGRAPHIES

Charlie Irawan Tan received his B.S. and M.S. degrees in Computer Science from National Dong Hwa University in 2004 and 2005, respectively. He is currently a Computer Science Ph.D. candidate in Computer Graphics at National Dong Hwa University. His research interests include steganography, animation, artificial intelligence, and game content generation.

Wen Kai Tai received his M.S. and Ph.D. degrees in Computer Science from National Chiao Tung University in 1989 and 1995, respectively. He is currently an associate professor of the Department of Computer Science and Information Engineering at National Dong Hwa University, Taiwan. His research interests include animation, real-time rendering, and gaming.