Generating Avoidance Motion Using Motion Graph

Masaki Oshita Naoki Masaoka

Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan
oshita@ces.kyutech.ac.jp masaoka@cg.ces.kyutech.ac.jp

Abstract. We propose a method of generating avoidance motions. We use a motion graph to generate continuous motions, including both avoidance and other kinds of motions. In the combat of real humans, trained fighters avoid an attack with minimal movement. To realize such avoidance motion, we developed criteria to find an appropriate path (series of edges) in the motion graph. The characters are expected to move their body by only a minimal distance to avoid an attack. We introduced attack, body and avoidance space–time volumes to evaluate this criterion. Each candidate path is evaluated according to the distance between attack and body volumes and the overlap between attack and avoidance volumes. We also introduced a method to control the execution speeds of edges, and thus adjust the timing of avoidance motions. Moreover, to find a path in real time, we developed methods to facilitate the searching process such as the use of grid-based indices to look up candidate paths and GPU-based quick collision detection to cull candidate paths. We tested our approach on an application in which a character avoids incoming balls controlled by a user and demonstrated the effectiveness of our approach.

Keywords: avoidance motion, motion graph, space–time volume.

1 Introduction

Generating the realistic animation of combating characters, especially the generation of avoidance motions, is a challenge in the field of computer animation, because a character's motion must vary dynamically in response to the opponent's motion. Currently, many computer games that involve combat between characters generate character motions by selecting a suitable motion from a set of a limited number of precreated motions and playing that motion. In the combat of real humans, trained fighters avoid an attack with minimal movement. However, this kind of avoidance motion cannot be realized in computer animation when taking the current approach. The characters in computer games instead avoid an attack by taking a large step or making a large leap. To realize good avoidance motions as real fighters do, the system requires methods for organizing many avoidance motions and executing an appropriate motion acceding to an incoming attack at an interactive speed.

In this paper, we propose a method to generate avoidance motions in real time by solving the abovementioned problems. We use a motion graph [2] to generate continuous motions, including both avoidance and other kinds of motion.

A motion graph is a set of connected short motion segments (edges). It is constructed from a set of long motion sequences by finding similar postures in the input motions and converting the similar postures into nodes of the motion graph and converting the motion segments into directional edges. Once a motion graph is constructed, a continuous motion is generated by traversing edges in the motion graph while playing them. However, to generate meaningful motion, the system needs rules for choosing an appropriate next edge or a path (series of edges) from the edge currently being played. Many kinds of rules have been proposed, such as those relating to walking [2,8], interactive control by the user [11], and reactions to impacts [1]. However, generating avoidance motion has been a difficult challenge.

In this research, we developed criteria to find an appropriate path (series of edges) in the motion graph to generate motion to avoid an incoming attack in a way similar to that employed by human fighters. The characters are expected to move their bodies only a minimal distance to avoid an attack. We introduced attack, body and avoidance space–time volumes to evaluate this criterion. Each candidate path (motion) is evaluated according to the distance between the attack and body volumes and the overlap between the attack and avoidance volumes. In addition, we also introduced a method to control execution speeds of edges, and thus adjust the timings of avoidance motions. Moreover, to find a path in real time, we developed methods to facilitate the searching process such as using grid-based indices to look up candidate paths and graphics processing unit (GPU)-based quick collision detection to cull candidate paths. We tested our approach on an application in which a character avoids incoming balls controlled by a user and demonstrated the effectiveness of our approach.

The rest of this paper is organized as follows. In Section 2, we review related works. In Section 3, we present the flow of our method. Section 4 explains the criteria for evaluating candidate paths. Section 5 explains the methods employed for computational efficiency. Finally, Section 6 presents experimental results.

2 Related Work

Motion graphs [2,3,6] have been widely used in recent research. To generate motions using a motion graph, criteria that determine appropriate edges and an efficient algorithm to search for a path (series of edges) that satisfies the criteria are necessary. Various criteria have been proposed depending on the types of motions, as mentioned in Section 1. To find a path, general algorithms such as the branch and bound algorithm [2], reinforcement learning [4], Markov decision process control [11,5], and the min-max algorithm [9] are used. However, these approaches cannot be simply applied to our problem, because a path cannot be evaluated until it reaches the avoidance motion part. It is thus difficult to search for avoidance motions efficiently.

Several research works have addressed the generation of combat or avoidance motion. Zordan et al. [13] employed a support vector machine, which is a pattern recognition technique, to select an avoidance motion according to feature vectors containing the position, direction and speed of an incoming attack. However, since the approach does not evaluate space–time conditions between the attack and avoidance motions, it is difficult to generate proper avoidance motion. In fact, a character

prepares for an incoming attack but then was hit by the attack. Lee and Lee [4] generated attack motions based on a given target position using a motion graph. However, to generate avoidance motions, not only a point in space and time but also space–time volumes must be considered. Therefore, generating avoidance motions is a more difficult problem. Shum et al. [9] selected combat motions including avoidance, but did not consider the criteria for natural-looking avoidance motions. Shum et al. [10] also generated combat motions using combined attack and avoidance motions. However, taking this approach, each avoidance motion is coupled with a corresponding attack motion and avoidance motions cannot be generated for any incoming attack. The patterns of avoidance and attack motions are limited. Wampler et al. [12] proposed a framework for planning two characters' continuous motions in real time. Their method also requires that the attacker's motion be generated using the same method, and avoidance motion for any attack is not realized. We propose novel criteria to select avoidance motions from a motion graph using space–time volumes.

3 Overview

Our system generates a character's avoidance motions using a motion graph that contains avoidance and other kinds of motions that are necessary for the application. Overall runtime processes are shown in Fig. 1 (a). When there is no incoming attack, any rules to select edges can be applied. When information of an incoming attack (its space–time volume) is given, our method searches for a path (series of edges in the motion graph) starting from the edge currently being executed to realize an avoidance motion. The system then executes the selected path to generate a resulting animation.

The details of the data representation of attack volumes are explained in Section 4.2. The space–time volume of an incoming attack can be generated in various ways depending on the application. In our experiment, we develop an application in which a user throws a ball at a character by clicking a point on the screen. In this case, an attack volume is generated from the half line in the virtual world corresponding to the clicked point on the screen. If a developer wants to generate an animation of fighting characters, he/she can generate the motion of an attacker using the motion graph or other dynamic motion-generation techniques. In this case, an attack volume can be generated from motion data with some additional information such as the body part used for the attack and the time at which the attack is supposed to hit.

In general, a large number of possible motions can be generated from a starting edge. For computational efficiency, a conventional search algorithm [2,3,4,5,9,11] attempts to cut less promising paths as early as possible and to develop promising paths by evaluating the incomplete candidate paths. However, we cannot apply the same approach for avoidance motion because a candidate path generally cannot be evaluated until it reaches the avoidance motion part. Therefore, our system enumerates possible candidate paths that include an avoidance motion first. The avoidance part of each candidate path is then evaluated to select the best among all candidate paths. This approach requires some computational time to enumerate and evaluate many possible candidate paths. For computational efficiency, we developed several methods that are explained in Section 5.

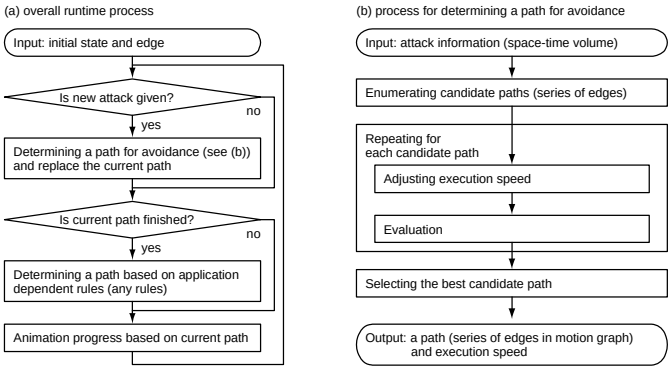

Fig. 1. System flow.

The process for determining a path to be executed is shown in Fig. 1 (b). When attack information is given, the system enumerates possible candidate paths starting from the current edge that have avoidance motion within a certain time window. Although we adjust the execution speed of each candidate path later, the timing of the avoidance motion is expected to be within a certain time window T_{window} from the timing of the attack. The interval from the beginning of the path to the center of the avoidance part t_{avoid} must satisfy the following condition.

$$t_{attack} - T_{window}/2 < t_{avoid} < t_{attack} + T_{window}/2, \quad (1)$$

where t_{attack} is the interval from the beginning of the path to the center of the attack time. In our implementation, we use $T_{window} = 1.0$. By traversing all edges from the current edge, all possible candidate paths are enumerated. The evaluation of candidate paths including calculation of the execution speed of them is explained in Section 4.

3.1 Constructing a motion graph including avoidance motions

To construct and represent motion graphs, we use methods similar to those used in previous works [1,2,3,8,9]. A motion graph consists of nodes and edges. Each directional edge represents a short segment of motion. Each node represents a posture connecting adjacent edges. By traversing edges, continuous motions can be generated. A motion graph is constructed from a number of motion clips by analyzing them and identifying similar parts in them and generating edges from the similar parts.

To realize avoidance motions, we label avoidance parts on edges in the motion graph. We specify avoidance parts (time intervals) in the original motion clips manually before constructing the motion graph; the information is inherited by the edges in the constructed motion graphs. During construction of the motion graph, each avoidance part is preserved and not divided into more than one edge.

In addition, some information is computed automatically. Body and avoidance volumes and speed adjustment condition are computed for each edge. A grid-based index is constructed for each node.

4 Evaluation of a candidate path

This section describes our method of selecting a path in the motion graph from the candidate paths to realize avoidance motion. As explained in Section 3, possible candidate paths are enumerated first. The execution speed for each candidate path is adjusted before each candidate path is evaluated.

4.1 Adjusting the execution speed

The timing of the avoidance motion is expected to match exactly with the timing of an incoming attack. However, in general, such paths hardly exist among the limited number of candidate paths. To address this problem, we adjust the execution speed of each candidate path so that the generated motion meets the timing constraint. For each candidate path, the execution speeds of edges in the candidate path are determined.

Our key insight for this process is that we change the execution speed for the part of the motion (edges in the motion graph) for which the change does not cause a noticeable problem. If part of the original motion is fast, it is considered that that part can be slowed down a little. On the other hand, if part of the original motion is slow, it is considered that that part can be sped up a little. If part of the original motion is still or almost still, it can be either slowed down or sped up, because neither greatly affects the original movement.

When the motion graph is constructed, the system analyzes each edge and determines if the edge can be sped up, slowed down, both, or neither on the basis of the velocities of the end effectors. The velocities of four end effectors (hands and feet) $v_{r\_hand}{}^{t}, v_{l\_hand}{}^{t}, v_{r\_foot}{}^{t}, v_{l\_foot}{}^{t}$ are calculated for each frame, and the maximum velocity among all frames and end effectors is denoted $v_{\max}$. If $v_{\max} > V_{fast}$, the movement of the edge is determined to be fast and allowed to slow down. If $V_{slow} > v_{\max} > V_{still}$, the movement of the edge is determined to be slow and allowed to speed up. If $v_{\max} < V_{still}$, the movement of the edge is determined to be still and allowed to either speed up or slow down. $V_{fast}, V_{slow}, V_{still}$ are thresholds. In our implementation, we set $V_{fast}, V_{slow}, V_{still}$ as 0.3, 0.1, and 0.0 m/s, respectively.

Each candidate path is adjusted so that its avoidance time and the attack time in the given attack information match. First, the overall scale of the speed change is calculated from the avoidance and attack times. Next, the speeds of edges that can be adjusted are scaled as shown in Fig. 2. For example, if the path must be sped up, the speed of all edges that are allowed to be sped up (dashed edges in Fig. 2) is scaled at the same ratio. We let t_{avoid} be the interval from the beginning of the path to the center of the avoidance part and t_{attack} be the interval to the center of the attack time window. The candidate path consists of adjustable edges and non-adjustable edges.

$$t_{avoid} = t_{avoid\_adjustable} + t_{avoid\_nonadjustable} \tag{2}$$

The time scaling parameter s_{time} for all the adjustable edges is computed as

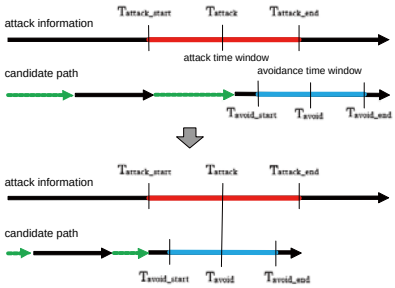

Fig. 2. Speed control.

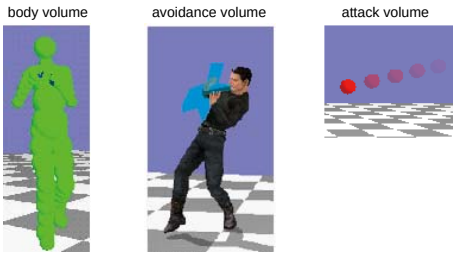

Fig. 3. Example of volumes.

$$s_{time} = \left(t_{attack} - t_{avoid\_nonadjustable}\right) / t_{avoid\_adjustable} \,. \tag{3}$$

If there is no edge that can be adjusted in a candidate path, the path is removed from the set of candidate paths. To avoid too much speed adjustment that may cause unnatural movement, we also evaluate the adjustment ratio of speed s_{time} as explained later. Our approach may result in discontinuity at transitions between scaled and non-scaled edges. To address this problem, smooth time warping between the edges or dynamic filtering ensuring continuous motion may be further applied.

4.2 Evaluation of a candidate path

This subsection describes our method of evaluating candidate paths. Even when the opponent character performs the same attack motion, appropriate avoidance motions in response to the attack vary depending on the relative positions and orientations of the characters and the defending character's current state. An appropriate candidate must be chosen on the fly. As explained in Section 1, human fighters attempt to avoid an attack with minimal movement. We developed criteria to find such candidates. A character is expected to move his/her body only a minimal distance to avoid an attack.

To realize such avoidance motion, obviously the character's body should not be touched by the attack. In addition, the character should move his/her body away from where an incoming attack passes through immediately before the attack would hit the body. To evaluate these factors, we introduce attack, body and avoidance space–time volumes. The factors are evaluated by computing the overlap or distance between volumes. The attack volume is given as the input to the system. The body and avoidance volumes are computed from the motion graph.

Human fighters sometimes shield or parry attacks by intentionally intercepting attacks with a body part. Our method does not consider such non-avoidance reactions to attacks and focuses on avoidance motions.

Definition of attack, body and avoidance volumes. Figure 3 shows an example of attack, body and avoidance volumes. These volumes are space–time (four-dimensional) volumes. The attack volume is the volume affected by the attack. The body volume is where the character's body exists. The avoidance volume is where the

character's body existed a while ago, but not now. The specific definitions of these volumes are given in the remainder of this subsection.

There are various ways of representing space–time volumes depending on the expected accuracy and efficiency. For our current implementation, we chose to use a set of spheres in discrete frames (1/10 seconds). We assign a number of spheres on the character's skeleton manually in advance with appropriate positions and radii so that the spheres cover the character's body. Using these preset spheres, the body volume in a frame is computed from the posture of the character. The positions of spheres between discrete frames are computed by interpolating two adjacent frames. During evaluation, overlap and distance between volumes are evaluated in discrete frames (1/20 seconds). The intervals between frames for volume representation and evaluation can be changed or be adaptive depending on the application. Especially when attacks are fast, smaller intervals may be necessary. The avoidance volume in each frame is defined as a difference between the body volume when the avoidance motion began and the body volume at the current time. An attack volume is given as an input. How to calculate attack volumes depends on the application.

Evaluation of the overlap between attack and avoidance volumes. The first criterion is the overlap between the attack volume and the avoidance volume of the candidate path. The penetration of the attack volume into the avoidance volume during avoidance motion is computed. Since we use a set of spheres to represent the volumes, we compute the sum of overlaps between each pair of spheres.

$$e_{avoidance}{}' = \sum_{t,i,j} \begin{cases} 0 & if\ D\left(S_{attack}{}^{t,i}, S_{avoidance}{}^{t,j}\right) \geq 0 \\ -D\left(S_{attack}{}^{t,i}, S_{avoidance}{}^{t,j}\right)\Delta t & if\ D\left(S_{attack}{}^{t,i}, S_{avoidance}{}^{t,j}\right) < 0 \end{cases} \tag{4}$$

for the *i*-th sphere of the attack volume and *j*-th sphere of the avoidance volume in frame t (interval of 1/20 seconds). $D(S_1, S_2)$ calculates the distance between/overlap of two spheres. If the value is positive, the spheres do not overlap and the value represents the distance between them. If the value is negative, the spheres overlap and the value represents the depth of penetration. Because the avoidance volume is defined as the difference between body volumes as explained above, the penetration depth in equation (4) is computed as follows, where t_0 is the time when the avoidance part starts.

$$D\left(S_{attack}{}^{t,i}, S_{avoidance}{}^{t,j}\right) = D\left(S_{attack}{}^{t,i}, S_{body}{}^{t_0,j}\right) - D\left(S_{attack}{}^{t,i}, S_{body}{}^{t,j}\right) \tag{5}$$

Finally, we scale this value from 0.0 to 1.0 using a scaling parameter $E_{avoidance}$.

$$e_{avoidance} = \sum_{t,i,j} \begin{cases} 1 - e_{avoidance}{}' / E_{avoidance} & if\ e_{avoidance}{}' > E_{avoidance} \\ 0 & if\ e_{avoidance}{}' \geq E_{avoidance} \end{cases} \tag{6}$$

In experiments, it appears that the character avoids the attack when $e_{avoidance}{}'$ is greater than 0.5. Therefore, we use $E_{avoidance} = 0.5$

Evaluation of the distance between attack and body volumes. The second criterion is the minimal distance between the attack volume and the body volume. It is computed as follows, where *t* is repeated for each frame of the candidate path.

$$d = \min_{t,i,j} \left\{ D\left(S_{attack}{}^{t,i}, S_{body}{}^{t,j} \right) \right\} \tag{7}$$

If $d < 0$, the attack and body volumes overlap and the character fails to avoid the attack. On the other hand, if $d > 0$, they do not overlap. In this case, the shorter the distance is, the better the avoidance motion looks. However, if we simply give high evaluation to shorter distances, the character tries to approach the attack, even when the character does not have to move to avoid the attack. Therefore, we decided to give a high evaluation to the shorter distance when the distance is greater than that when the motion starts d_0, where D_{body} is a scaling parameter.

$$e_{body} = \begin{cases} 1 & if\ d \le d_0 \\ \left(d - d_0\right) / D_{body} & if\ d_0 < d < d_0 + D_{body} \\ 1 & if\ d \ge d_0 + D_{body} \end{cases} \tag{8}$$

Evaluation of the rate of adjustment of the execution speed. In addition to the two criteria above, the rate of adjustment of the execution speed is evaluated because it may cause unnatural motions. Although the allowable speed change depends on the motion, through our experiments, we determined that it is acceptable if the speed rate change is within a factor of 4 $\left(E_{time} = 4\right)$.

$$e_{timescale} = \begin{cases} \left|s_{time}\right| / E_{time} & if\ \left|s_{time}\right| > 1 \\ 1 / \left(\left|s_{time}\right| E_{time}\right) & if\ \left|s_{time}\right| < 1 \end{cases} \tag{9}$$

Total evaluation. Each candidate path is evaluated according to

$$e = w_a e_{avoidance} + w_b e_{body} + w_t e_{timescale}, \tag{10}$$

where w_a, w_b, w_t are weights that control the contributions of these factors, while $E_{avoidance}$, D_{body}, and E_{time} in equations (6)–(9) are determined considering each factor independently. In our implementation, we set all weights w_a, w_b, w_t as 1.0. As explained in Section 3, among all candidate paths, the candidate path whose evaluation value is smallest is chosen and used to generate avoidance motion.

5 Methods employed for computational efficiency

Because the number of candidate paths can be large, enumerating and evaluating all candidate paths is time consuming. Therefore, we introduce a grid-based index and

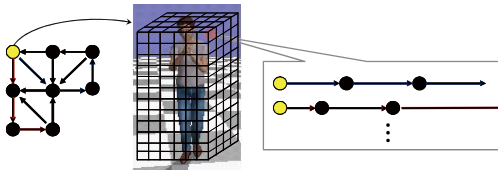

Fig. 4. Grid-based index.

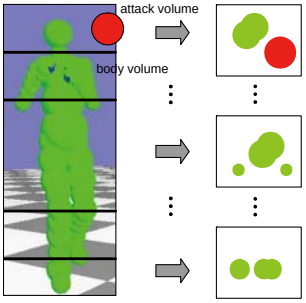

Fig. 5. GPU-based collision detection.

GPU-based collision detection. These are popular approaches to achieve computation efficiency. This section describes how we employ these approaches in our system.

5.1 Grid-based index for candidate paths

The most important criterion for selecting candidate paths is the overlap between attack and avoidance volumes. The number of candidate paths having overlap is limited compared with the total number of possible candidate paths. Therefore, we index such candidate paths using a grid-based index for each node in the motion graph. Without traversing the motion graph from the current edge, the candidate paths can be retrieved from the index of the current node according to the attack volume. If there is no candidate path for which the avoidance volume in the index, candidate paths are enumerated by traversing the motion graph as explained in Section 3.

The grid-based index is constructed in advance for each node in the motion graph. The space around the character at the node is divided into a grid as shown in Fig. 4. In our implementation, the size of each cell in the grid is 10 cm. First, possible paths that start from the node and contain avoidance motion within a certain time window are enumerated by traversing the motion graph. Then, for each cell, all paths for which the avoidance volume overlaps the cell are recorded.

During runtime, the attack volume is rotated and transformed into the local coordinates of the current node. Corresponding cells in the grid of the current edge are then determined according to the attack volume. The number of corresponding cells can be more than one since the attack volume is a space–time volume.

5.2 GPU-based collision detection for culling

Another important criterion for selecting candidate paths is the overlap between attack and body volumes. If they overlap on a candidate path, it means that the attack hits the character and the candidate paths should be removed. However, detecting the overlap between two volumes is time consuming. Therefore, we introduce GPU-based quick collision detection to cull candidates.

We divide the space into several horizontal segments as shown in Fig. 5. For each segment, collision detection between volumes is computed on a two-dimensional

space by drawing volumes. In our implementation, we divide the space into nine horizontal segments. For each segment of each edge in the motion graph, the place where the body volume exists during for entire edge is drawn on a texture in the local coordinates in advance. During runtime, the attack volume is first drawn on a texture for each segment. The body-volume texture of each segment is then drawn on the same texture by applying rotation and translation according to the character's position and orientation when the edge is executed. The number of pixels of the overlap between attack and body volumes is counted using the GPU. When the number is greater than a threshold, it is expected that the attack hits the body.

Since we discretize the space and time, this is an approximation method for collision detection. However, it is considered to be sufficient for quick culling.

6 Experiments

We implemented the proposed methods and developed an application in which a user throws balls at a character by clicking on the screen and the character performs an avoidance motion. Although our method can be used to generate animations of multiple characters combating each other, we developed this application of generating the animation of a single character avoiding given attacks so that we can test our methods with various inputs of attacks. When there is no attack coming, the next edge is determined randomly. When an attack is given by the user, an attack volume is created from the trajectory as explained in Section 3. Only the part of the trajectory of the ball that is close to the character is used to create the attack volume. An avoidance motion is then generated according to the attack volume. When there no valid candidate path is found, the character may be hit by the ball.

For our experiments, a motion graph was constructed from motion clips of 2 minutes and 40 seconds including 16 avoidance motions such as twisting the upper body, ducking, swinging at the waist, and jumping. The experiments were done on a standard PC with a Pentium 4 CPU, 2.5 GB RAM, and GeForce FX5700 256M GPU.

We also implemented multi-threading to execute and search motions. When an attack is given, the process of finding a path for avoidance motions begins on the background thread. Until the path is found, the next edge is randomly selected when the current edge has finished being executed. The candidate paths that do not contain the next edge are removed from the candidates. When a path is determined, the background thread stops and the path is used in the thread for motion execution.

Table 1. Average time for selecting a candidate path.

Condition		Time (milliseconds)
With GPU-based culling	Candidate paths in the grid	14
	No candidate path in the grid	225
Without GPU-based culling	Candidate paths in the grid	18
	No candidate path in the grid	496

6.1 Computational time for path selection

We measured the computational times from the time of an attack given by the user until the time that a path for avoidance motion is selected. In this experiment, we did not employ multi-threading. The measured time is that for enumerating and evaluating candidate paths.

The results are presented in Table 1. As explained in Section 5.2, when there are candidate paths in the grid-based index, these paths are evaluated. On the other hand, when there is no candidate path, candidate paths are enumerated. It was about 20 times faster to use the grid-based index, because it does not require the enumerating of candidates and there are fewer candidates. The average number of candidate paths was 5 when the grid-based index was used, while the number was 100 when the grid-based index was not used. This explains the difference in results for the two cases.

In our experiments, in 90% of cases, the grid-based index was used. This is probably because the user attempted to attack the character and clicked a point on or near the character. If an attack volume is away from the character, it is likely that no candidate is found in the grid-based index. However, in general, it is expected that attacks occur near a character. Employing GPU-based collision detection for culling, the computation was about twice as fast on average. This was effective especially when there was no candidate path in the grid-based index.

The overall computational was approximately 20 to 200 milliseconds. Our method can be used in real-time applications employing multi-threading even when the index is not used. When we activate multi-threading, in 50% of cases, a path was found before the current edge was finished being executed.

6.1 Evaluation of avoidance motions

We also evaluated the quality of generated avoidance motions. In general, it is difficult to evaluate the naturalness of motion. In our experiment, we asked a subject who is a graduate student majoring in computer animation to observe all candidate paths and to select the one that generates the most natural-looking avoidance motion. We then compared whether the path selected by the subject matched the path selected using our method. In 90% of 40 trials, the paths matched. When they did not match, the major reason was that there were no good avoidance motions in the candidate paths and selecting the best path was difficult even for us. This is because the number of reachable avoidance motions in the motion graph within the time window was limited even though it seemed that a sufficient number of avoidance motions were used. We could use a wider time window. However, that may increase the computational time and require large speed adjustments. We constructed a standard motion graph in this research. We may need to develop a method to construct a sophisticated motion graph with which various avoidance motions can be easily reached from any node. This is one of our future works. Conducting extended evaluation including subjective evaluation by many subjects, comparison with a ground truth (motion capture data of combating people), and measuring the rate of successful avoidance are other future work.

7 Conclusion

We presented a method of generating avoidance motions using a motion graph. We proposed new criteria based on attack, avoidance and body space–time volumes. We also introduced methods to achieve computational efficiency. In addition to the method of constructing a sophisticated motion graph as mentioned in Section 6, development of data structures and algorithms that are more efficient such as [5] is future work. In general, it is difficult to select a perfect avoidance motion for any attack. We may need a method of modifying a selected motion according to an attack such as [1]. As explained in Section 4.2, human fighters sometimes shield or parry attacks by intentionally intercepting attacks with a body part instead of avoiding them. Generating these kinds of motions is also a future work.

References

1. Okan Arikan, David A. Forsyth, James F. O'Brien. Pushing People Around. ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005, pp. 59-66, 2005.
2. Lucas Kovar, Michael Gleicher, Frederic H. Pighin. Motion Graphs. ACM Transactions on Graphics (SIGGRAPH 2002), 21(3), pp. 473-482, 2002.
3. Jehee Lee, Jinxiang Chai, Paul Reitsma, Jessica Hodgins, Nancy Pollard. Interactive Control of Avatars Animated with Human Motion Data. ACM Transactions on Graphics (SIGGRAPH 2002), 21(3), pp. 491-500, 2002.
4. Jehee Lee, Kang Hoon Lee. Precomputing Avatar Behavior from Human Motion Data. ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pp. 79-87, 2004.
5. Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, Zoran Popović. Motion Fields for Interactive Character Locomotion. ACM Transactions on Graphics (SIGGRAPH Asia 2010), 29(6), Article No. 138, 2007.
6. Yan Li, Tianshu Wang, Heung-Yeung Shum. Motion Texture: A Two-Level Statistical Model for Character Motion Synthesis. ACM Transactions on Graphics (SIGGRAPH 2002), 21(3), pp. 465-472, 2002.
7. James McCann and Nancy S. Pollard. Responsive Characters from Motion Fragments. ACM Transactions on Graphics (SIGGRAPH 2007), 26(3), Article No. 6, 2007.
8. Paul S. A. Reitsma, Nancy S. Pollard. Evaluating Motion Graphs for Character Navigation. ACM SIGGRAPH/Eurographcis Symposium on Computer Animation, pp. 89-98, 2004.
9. Hubert P. H. Shum, Taku Komura, Shuntaro Yamazaki. Simulating Competitive Interactions Using Singly Captured Motions. ACM Virtual Reality Software and Technology 2007, pp. 65-72, 2007.
10. Hubert Shum, Taku Komura, Masashi Shiraishi, Shuntaro Yamazaki, Interaction Patches for Multi-Character Animation. ACM Transactions on Graphics (SIGGRAPH Asia 2008), 26(3), Article No. 114, 2008.
11. Adrien Treuille, Yongjoon Lee, Zoran Popović. Near-optimal Character Animation with Continuous Control. ACM Transactions on Graphics, 26(3), Article No. 7, 2007.
12. Kevin Wampler, Erik Andersen, Evan Herbst, Yongjoon Lee, Zoran Popović. Character Animation in Two-Player Adversarial games, ACM Transactions on Graphics (SIGGRAPH 2011), 29(3), Article No. 26, 2010.
13. Victor Zordan, Adriano Macchietto, Jose Medin, Marc Soriano, Chun-Chih Wu, Ronald Metoyer, Robert Rose. Anticpation from Example. ACM Virtual Reality Software and Technology 2007, pp. 81-84, 2007.