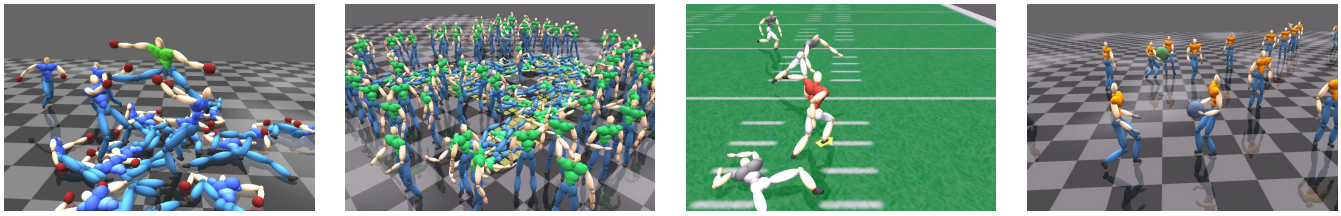
Interaction Patches for Multi-Character Animation

Hubert P. H. Shum* Taku Komura†
Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

Masashi Shiraishi‡
Graduate School of Engineering
Waseda University

Shuntaro Yamazaki §
Digital Human Research Center
National Institute of Advanced
Industrial Science and Technology



(a) (b) (c) (d)

Figure 1: Multi-character animations are synthesized from single-person Motion Capture data. The individual interactions between nearby characters are precomputed into *interaction patches* by expanding game trees during the off-line processing stage. Our system automatically concatenates the patches and generates a complex multi-character animation, such as (a) one person fighting with many enemies, (b) a group of characters falling down onto each other like dominos, (c) an American football player holding a ball and escaping from tackling defenders, and (d) a group of people passing luggage from one to another.

Abstract

We propose a data-driven approach to automatically generate a scene where tens to hundreds of characters densely interact with each other. During off-line processing, the close interactions between characters are precomputed by expanding a game tree, and these are stored as data structures called *interaction patches*. Then, during run-time, the system spatio-temporally concatenates the interaction patches to create scenes where a large number of characters closely interact with one another. Using our method, it is possible to automatically or interactively produce animations of crowds interacting with each other in a stylized way. The method can be used for a variety of applications including TV programs, advertisements and movies.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Character Animation, Human Motion, Crowd Simulation

1 Introduction

Scenes of battlefields, panicked crowds and team sports in movies and TV programs involve a huge number of interactions of multiple characters. Existing methods have problems creating such interactions. Manually composing the scene using singly captured motions or keyframed motions requires a huge amount of labor by the animator. Flocking-based methods [Reynolds 1987; Helbing et al. 2000] have problems simulating close interactions that involve a lot of kinematic constraints. Previous optimization-based methods [Lee and Lee 2004; Treuille et al. 2007; Shum et al. 2008] suffer when creating artistic interactions, as the objective functions are designed just to benefit each character.

When we watch fighting scenes in movies, we immediately realize that there are a variety of interactions appearing stylized; artistic and logically clear as if they are designed by an artist. At the same time, we also realize that the patterns of interactions are very simple. For example, in a scene where a main character fights with many background characters, most interactions between them follow the rule of "background character: attack", "main character: avoid", "main character: counter attack" and "background character: knocked down".

This observation leads us to develop an algorithm that is flexible enough for the user to design his/her favorite interaction, while sufficiently automated so that the user can create a large-scale animation involving a number of characters with the least effort. Our system simulates the minimal unit of interactions between two characters based on abstract instructions given by the user, and stores the result as structures called interaction patches. The interaction patches are spatio-temporally concatenated to compose a large-scale scene in which the characters interact with each other, such as one person fighting with many enemies (Figure 1a), a group of characters falling down onto each other like dominos (Figure 1b), an American football player holding a ball and escaping from tackling defenders (Figure 1c) and a group of people passing luggage one to another (Figure 1d).

1.1 Related Work

Simulating close interactions of multiple characters has been attracting researchers due to the high demand in movies and computer games. When creating an animation of a large-scale scene where the characters have close interactions with one another, it

*e-mail: hubert.shum@ed.ac.uk
†e-mail: tkomura@ed.ac.uk
‡e-mail: s-masashi331@moegi.waseda.jp
§e-mail: shun-yamazaki@aist.go.jp

ACM Reference Format
Shum, H., Komura, T., Shiraishi, M., Yamazaki, S. 2008. Interaction Patches for Multi-Character Animation. *ACM Trans. Graph. 27*, 5, Article 114 (December 2008), 8 pages. DOI = 10.1145/1409060.1409067 http://doi.acm.org/10.1145/1409060.1409067.

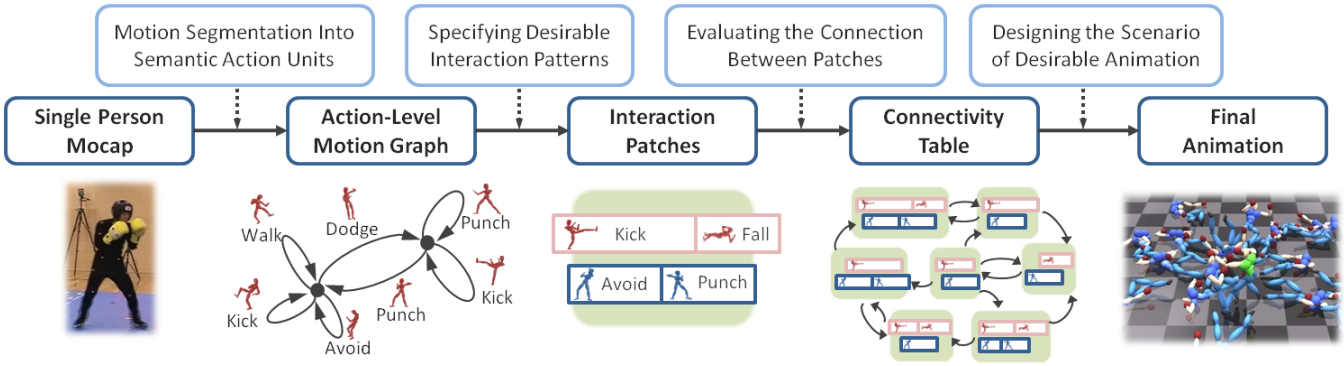


Figure 2: The outline of the proposed method.

can be done with either (1) a top-down approach that models the overall movements of all the characters first and then further models the details, or (2) a bottom-up approach where the individual interactions are modeled first and the overall scene is composed by assembling them.

Crowd simulation, which is a top-down approach, provides scalable methods to generate motions of multiple characters moving in the scenery. For example, Sung et al. [2004] propose the use of a probabilistic model for the selection of actions. Treuille et al. [2006] use continuum dynamics to determine the flow of people moving in the space. In these works, the interactions between the characters are rather simple, such as avoiding other pedestrians or walking along with another character. No close interactions such as holding or strongly pushing away others are included. Helbing et al. [2000] propose a dynamical model to simulate the movements of people in panic, which is similar to the flocking [Reynolds 1987] model. However, in their work, there is no model to convert the motions of particles to human actions when close contacts are involved. A bottom up approach, which is to build the individual interactions first, and combine them together to design the whole scene, is more suitable for our objective. The following approaches can be categorized as bottom-up.

Data driven approaches: Capturing the interactions of multiple persons with a motion capture system, and using machine learning techniques to produce motions under different situations [Park et al. 2004; Lee et al. 2007] is a straight forward approach. Park et al. [2004] create animations of two persons playing Tae Kwon Do and dancing by using Hidden Markov Model. In this method, because of the limitation of motion capture, the Tae Kwon Do players cannot seriously hit each other, and only the combinations of actions that have been captured can be replayed. Lee et al. [2007] record group behaviors with an overhead camera and model how individuals move in the crowd. Such approaches are difficult to apply to motions involving close interactions. Kwon et al. [2008] propose a method to create and edit the movements of characters in a group. The only interaction they handle is avoiding collisions between close characters.

Response motions when a person is being pushed, pulled or hit have become attractive to researchers due to their high demand in video games. Zordan et al. [2005] simulate the response motion after being attacked using rag-doll physics and motion capture data. Arikan et al. [2005] use machine learning techniques for training the system to produce plausible reactions. Synthesis of reactive motions against obstacles has also been explored by frame-based optimization techniques [Abe et al. 2007], support vector machines [Zordan et al. 2007] and spacetime constraints [Liu et al. 2006]. These methodologies are useful for designing individual short interactions between two characters. However, when a large-scale scene is to be created, we must allocate the characters in the scene and plan their movements.

Optimization-based methods can be one solution to this problem. Lee and Lee [2004] precompute the optimal policy for the boxer character to approach and hit a target. They focus mainly on relatively simple interactions, such as a single punch, due to the high dimensionality of the state space, in which each point defines a unique condition to select an action. Treuille et al. [2007] tackle this problem by using a near-optimal approach; the objective function is represented by a weighted sum of bases functions, and the policy is optimized by recursively updating the weights. They successfully created motions of pedestrians avoiding each other, but have not produced close interactions such as those handled in this paper. Shum et al. [2008] cope with the problem of high dimensionality by collecting samples in the state space where there are meaningful interactions. The problem of these learning-based methods is that a huge number of samples are required to obtain a satisfactory result. They also have problems simulating stylized interactions as the objective functions are designed for each character just to compete well. Game tree expansion [Shum et al. 2007] is an effective method to synthesize realistic interactions among characters. This method is similar to the way that computer-based chess players select their moves. The problem is that this method requires an exponential amount of computation.

1.2 Our Approach

Our work is inspired by the idea of Motion Patches [Lee et al. 2006], where the large-scale scene is composed of building blocks. Using their approach, it is possible to generate an animation where the characters interact with the environment. However, it is not possible to generate an animation where multiple characters densely interact with each other. In this paper, we precompute the complex interactions of multiple characters and use them as the building blocks to compose the final scene.

The outline of our approach is shown in Figure 2. It is composed of five steps: (1) Capture the motion of a single person using a motion capture system. (2) Create the action-level motion graph [Shum et al. 2007], in which the actions are all annotated. (3) Compose the set of minimal units of interactions, which we call the interaction patches, by specifying the pattern of interactions and expanding the game tree. These three steps are explained in Section 2. (4) Generate two tables that list how each interaction patch can be temporally and spatially concatenated with other interaction patches to compose large-scale scenes. This process is explained in Section 3. The processes up to here are done offline. (5) Compose a scene by concatenating the interaction patches. This is the only online process, which allows the user to optionally give high-level commands and see what they can get immediately. The details are explained in Section 4.

Contribution

1. We propose a method to synthesize realistic interactions between characters by expanding the game tree, based on the pattern of interactions specified by the user. Since the pattern is specified, the number of combinations is small, and we can obtain realistic interactions with a limited amount of computation. These interactions are saved as interaction patches to be used during runtime.

2. We propose a new algorithm to synthesize a large-scale scene in which the characters densely interact with each other. The precomputed interaction patches are spatio-temporally concatenated to compose a large-scale scene.

2 Interaction Patches

The interaction patch is composed of the initial condition of the two characters and the list of short motion clips performed by each of them. The initial condition includes the distance between the two characters (r), the relative orientation of each character with respect to the other (θ^1 and θ^2), and the delay in either of the characters to start the first action (t_{diff}).

In the rest of this section, we first explain how we preprocess the motion capture data, and then explain how the interaction patches are generated. Finally we explain how they are evaluated.

2.1 Preprocessing Motion Data

A motion capture system is used to obtain a long sequence of motion data performed by a single person. We assume the motion data is preprocessed and stored as an action-level motion graph [Shum et al. 2007]. This is a motion graph structure in which the nodes represent postures to start or end actions and the edges represent semantic actions. The list of annotations used in this research are shown in Table 1. An example of an action-level motion graph is shown in the second left image of Figure 2. The readers are referred to [Shum et al. 2007] for further details.

2.2 Composing Interaction Patches

The process of composing interaction patches is to let the user specify the pattern of actions, sample the initial condition of the two characters and simulate the interactions between them. An overview, showing the composition of an interaction patch is shown in Figure 3. Each process is explained in the following subsections.

Specifying Pattern of Interactions: A user first gives a list, defined here as a *PatternList*, that describes the pattern of the interaction between two characters: $PatternList = \{ (CharID_1, Annotation_1), ..., (CharID_n, Annotation_n) \}$, where $Annotation_i$ is the annotation embedded in the action-level motion graph, $CharID_i$ is the identity of the character who performs this action, which is either 1 or 2, and n is the total number of actions in the pattern. In our system, multiple actions may share the same annotation. Therefore, an annotation represents a cluster of actions, rather than a specific action. Figure 3 (upper left) shows an example of *PatternList*. It should be noted that the list defines only the starting order of the actions, and does not mean each character has to wait for the other character to finish its action to start a new action.

Table 1: The table of annotations used in this research

Scene	Annotations
Fight	punch, kick, avoid, dodge, transition, falling
Football	run, jump, avoid, tackle
Mouse	avoid, pushed
Crowd falling	falling
Luggage carry	carry, walk, hand, receive, turn

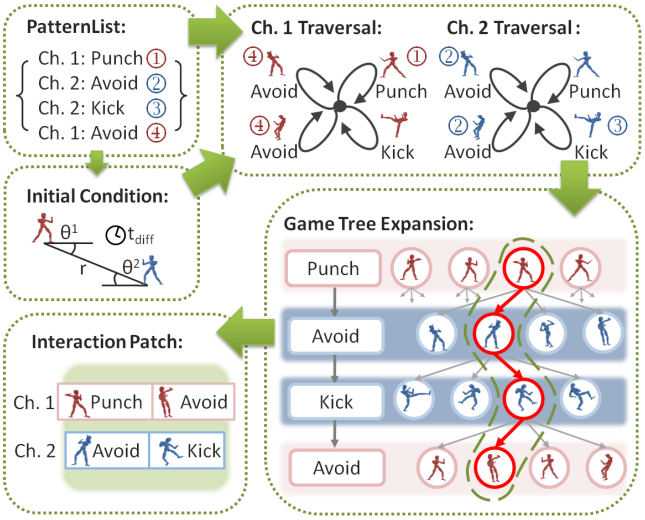


Figure 3: Given the *PatternList* (upper left), the system sets the initial condition (middle left). Using these data, the action-level motion graphs are traversed by both characters (upper right). The traversal process is equivalent to expanding the game tree (lower right) as there are multiple choices for the same annotation. The good interactions are stored as interaction patches (lower left).

Sampling Initial Conditions: Once the pattern of interaction is determined, the initial conditions of the characters are sampled based on the annotation of first actions for each character (Figure 3, middle left). For most of the actions, there is a range in the initial condition parameters r, θ^1, θ^2, t_{diff} when the action becomes successful. For attacks or tackles, the other character must be in the front at some distance and the valid range is relatively narrow. On the other hand, avoiding actions are valid as far as the character can get away from the opponent, which means the range is larger. We predefine the valid range of each parameter for each annotation. The system computes the intersection of the valid range for the characters' first actions, and performs uniform sampling in the intersection. In our system, distance is sampled every $20cm$, angles are sampled every $20°$, and time difference is sampled every $0.1s$.

Expanding Game Tree: When simulating the interaction between the two characters, each character is controlled by its own action-level motion graph. Starting from the sampled initial condition, each character traverses its own action-level motion graph according to the pattern of annotations given by the *PatternList* (Figure 3, upper right). As the annotation represents a cluster of actions, we have multiple choices of actions for each annotation. Since *PatternList* contains a list of annotations, there are exponential combinations of instances per *PatternList*. The process to evaluate all possible combinations is equivalent to expanding a game tree (Figure 3, lower right). In this game tree, each node represents an action to be launched by the corresponding character, and each edge directs the subsequent action by either character.

When expanding the game tree and evaluating the sequence of actions, some combinations are considered invalid for the following reasons:

- **Invalid distance:** We avoid interactions in which the characters stand too close, as they can cause serious penetrations.

- **Incorrect order of actions:** As the duration of each action is different, sometimes the overall order of the actions does not coincide with the pattern; such series of actions are discarded.

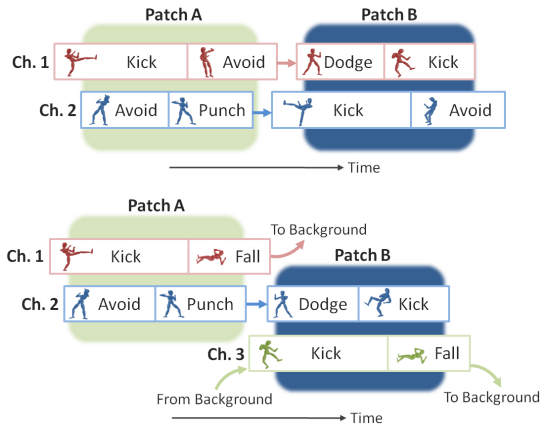


Figure 4: Two cases of temporal concatenation of interaction patches. Two characters finishing the previous interaction patch rejoin in the next patch (upper). One character starts to interact with a different character after finishing the previous patch (lower).

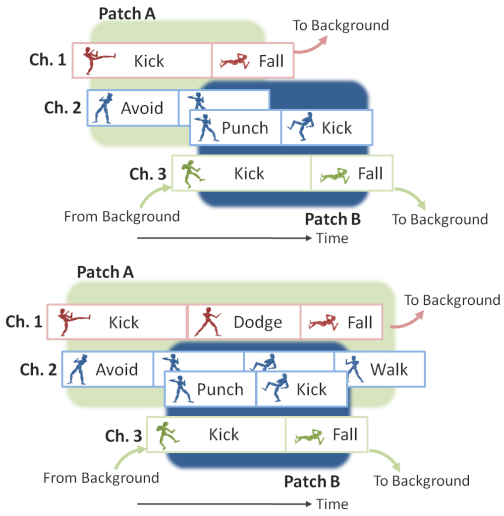


Figure 5: The condition for applying the spatial concatenation to the interaction patches: Either the series of actions in the initial and final part of the patches must overlap (upper) or the whole series of actions of one interaction patch overlaps with part of the other interaction patch (lower).

Close interactions involve a lot of close contacts of body segments. We need to evaluate whether the segments collide or not. We represent the body segments with rectangular bounding boxes and check if any segments are overlapping. If the colliding segment has large linear / angular momentum, response motion of being pushed or falling down is immediately launched. We compare every posture of the response motion with the posture at the moment when the impulse is added to the body. The best matching frame is used as the starting frame of the response motion [Zordan et al. 2005]. If the segments unintentionally collide, such as when a character is supposed to successfully avoid the attack according to the given pattern but gets hit, this sequence of actions is discarded.

2.3 Evaluating the Interactions

After expanding the game tree, we evaluate the interactions using a cost function. Any paths connecting the root and leaf nodes of the game tree form a series of interactions between the two characters. The set of interactions with a score above a threshold are stored as interaction patches. The design of the evaluation scheme is specific to the type of interactions. We used the linear combination of the following objective functions in our experiments.

- **Contact criterion:** For some actions such as holding the hand, punching the face, and tackling the body of the other person, some parts of the bodies must contact either for a moment or throughout the timeline. Better scores are given to a series of actions that result in desired contacts.

- **Relative distance/orientation criterion:** For actions such as dancing, the characters need to stay close and face each other for some period. Similarly, for interactions such as one character punching and the other avoiding, the defender should get away from the punch, but needs to face the attacker while avoiding it. For these interactions, there are desired distances and relative orientations of the root of the body at some moment / throughout the motion. We can evaluate the interactions based on the difference of the resulting values and the desired values.

- **Timing criterion:** Some combinations of actions performed by both characters need to be well synchronized. We consider those interactions with small timing differences to be better.

All the interactions designed in our experiments are modeled by different combinations of the above functions. The blending ratio are manually tuned for each example.

2.4 Computational Efficiency

Since the process of constructing the interaction patches involves game tree expansion, the computational cost is of an exponential order. In general, when fully expanding the game tree to evaluate the interactions of characters, the computational cost is A^D, where A is the average number of available actions, and D is the depth of the tree to be expanded. However, we can greatly reduce the cost by making use of the following features:

1. **As the patterns of actions are given**, the number of actions to be expanded at each level are much fewer than that of doing a full search. Assuming the actions are evenly divided into N types of annotation, the computational cost will be reduced to $(\frac{A}{N})^D$. At the same time we can get high quality samples, as the pattern of interaction is a very important factor to determine the realism of the interaction.

2. **As the *PatternList* is short**, the depth of the expanded tree, D, is limited. This is because only short interaction patches are required in our system. We can generate longer interactions, and those of more than two characters, by concatenating the interaction patches based on the method explained later in Section 3.

3 Connecting Interaction Patches

We compose large scale scenes by connecting the interaction patches. Long series of interactions can be created by temporally concatenating the interaction patches. Animations of more than two characters concurrently interacting can be composed by spatially concatenating the interaction patches. We check if such concatenations are possible for every pair of interaction patches, and save this information in a table. The details of checking the eligibility of temporal and spatial concatenations are explained in the following subsections.

3.1 Temporal Concatenation of Interaction Patches

Two interaction patches A and B can be temporally concatenated if (1) both of the characters finishing patch A start interacting again in patch B (Figure 4, upper), or (2) one of the characters finishing patch A joins patch B and starts to interact with a different character (Figure 4, lower).

The patches must satisfy two further conditions to be temporally concatenated: Firstly, the motions when switching from patches A to B must be continuous; this can be examined by checking the continuity of actions in the motion graph. Secondly, if the characters in the two patches are different, as in Figure 4 (lower), we must make sure the leaving character in patch A does not collide with the joining character in patch B. The leaving character either leaves the scene or joins another interaction patch with another character. For example, in Figure 4 (lower), after patch A, character 1 goes away and character 3 joins in patch B. Collision detection based on the two bounding boxes that surround character 1 and character 3 is carried out for all actions in the patch. Only if there is no collision can patch A and B be temporally concatenated.

3.2 Spatial Concatenation of Interaction Patches

The animator might need a scene where more than two characters concurrently interact; we can compose such a scene by spatially concatenating interaction patches of two characters. For example, the animator might need a scene in which a football player jumps up and avoids tackles from two opponents, one from the left and another from the right. This scene can be composed using two interaction patches, in which (1) a character jumps and avoids the tackle from the left, and (2) a character jumps and avoids the tackle from the right. There are two conditions for such a concatenation (Figure 5). First, the two uncommon characters in the two patches (character 1 and 3 in Figure 5) must not collide into each other. This condition is the same as the one in temporal concatenation. Second, the common character in the two patches (character 2 in Figure 5) must conduct the same series of actions for a continuous duration. The duration of overlap does not have to cover the whole interaction patch. If the ending part of one patch and the initial part of another patch overlap (Figure 5, upper) or if the whole series of actions in the shorter patch completely overlaps with a part of the longer patch (Figure 5, lower), this condition is satisfied.

4 Scene Composition

Once we know which interaction patches can be concatenated, we can automatically compose large-scale scenes by spatio-temporally concatenating the patches. In this section, we explain the process of composing the scene: First, we explain the criteria for selecting the next interaction patch among all the available ones, and then explain how these criteria are applied to generate the scene. Finally, we explain how to reuse characters that exited interaction patches for other interaction patches later in the scene.

4.1 Selecting Patches

Among all the patches that can be connected to the currently played one, our system excludes those which result in collisions, and then selects the best one among the rest based on an objective function explained in this subsection.

First, we exclude the patches that result in collisions. If a patch requires the system to add a new character to the scene, we need to ensure that the newly added character does not collide with any other characters present in the scene. This is done by representing each character as a bounding box and checking if the new character overlaps with those in the scene. Then, we evaluate the interaction patches based on the following factors:

- **Density of characters:** Because there are going to be a large number of characters involved in the interactions, we favor patches that allocate characters in open space. This is evaluated as follows:

$$s_d(p) = \frac{1}{d_p + 1}$$

where d_p is the current density of characters at the region where the candidate interaction patch p will occupy.

- **Frequency of the usage:** As we prefer the characters not to keep repeating similar movements, lower scores are given to patches which have been recently used. We define a parameter f_p to represent the usage of the patch p; once a patch is used, its corresponding f_p value is increased by one. On the other hand, the value is decreased by 10% each time other patches are selected. The usage score of the patch is calculated as follows:

$$s_f(p) = (1 - min(f_p, 1))$$

- **User preference:** We provide a simple interface for the user to select the preferred type of actions represented by action annotations. The patches that include such types of action are given better scores: $s_u(p) = 1$ if the action satisfies the user's preference and $s_u(p) = 0$ if it does not.

The final score of a patch is defined as the weighted sum of the above factors:

$$S(p) = w_d s_d(p) + w_f s_f(p) + w_u s_u(p) \quad (1)$$

where p is the patch to be evaluated, w_d, w_f, w_u are the weights for each factor, which we set as $w_d = 10, w_f = 1000$ and $w_u = 10000$. The patch that returns the highest score is selected.

4.2 Concatenating Interactions

Here we explain how to generate scenes of continuous interactions involving many characters by concatenating the interaction patches.

When an interaction patch is about to end, we automatically select the patch that can be temporally concatenated by evaluating all the connectable patches using Equation (1). If there are any patches which are spatially connectable, such patches are also evaluated by Equation (1) and the one with the best score is concatenated.

Then, the movements before and after the interaction for characters are generated by a locomotion engine which controls the character in a greedy manner. The locomotion engine selects a movement which is collision free and transfers the character as close as possible to the target position. The movements of the characters are determined backward and forward in time starting from the moment of the interaction. For those characters that appear from the background, the starting point is set at a location outside the scene in the radial direction. The motions of the character whose interaction happens first is decided first. Therefore, when deciding the locomotion of each character, we only need to avoid the the characters that are already in the scene. Although more elaborate locomotion engines based on model predictive control [Lau and Kuffner 2005] or reinforcement learning [Lo and Zwicker 2008] might perform better, our controller works well for the scenes we simulated.

An example of an overall time line is shown in Figure 6 (upper), in which character 1 (Ch.1) interacts with character 2, 3, 4 and 6 (Ch.2, Ch.3, Ch.4 and Ch.6) with temporal concatenation. The interaction patch shared by Ch.1 and Ch.4 is spatially concatenated with another patch shared by Ch.4 and Ch.5. A corresponding fighting scene is shown in Figure 7. Ch.1 (blue) first attacks Ch.2 (green) at the right side of the image, and next Ch.3 (grey) at the top, then Ch.4 (violet) at the left, and finally Ch.6 (orange) at the bottom. When Ch.4 falls down, this motion is spatially concatenated with another interaction patch, in which it falls over character Ch.5 (cyan). Once the interaction patches are fixed, the motions of the characters entering the scene are decided.

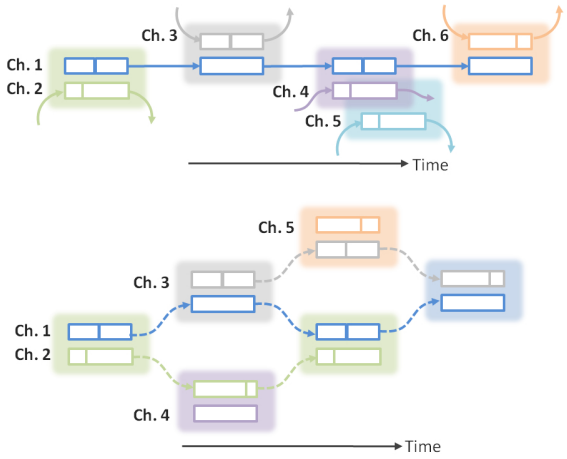


Figure 6: The structure of scenes composed by our method. A main character interacts with many background characters (upper). With characters recycled, they can continuously interact with other characters (lower). The dotted lines indicate that adjustment motions may be required to connect two patches.

4.3 Recycling Characters

When multiple characters continuously interact, they need to repeatedly enter and exit interaction patches (character 1 to 3 in Figure 6, lower). For instance, if we want to design such a scene for two characters, both characters going out from a patch need to rejoin in the next patch. However, sometimes these kind of patches cannot be found due to the distinct initial condition to start an interaction patch. We solve this by giving the characters the degrees of freedom to adjust their locations, orientations and postures.

First, we introduce the concept of standard pose, which is a pair of postures for two characters, from where the two characters can easily find ways to enter various interaction patches (Figure 8). This corresponds to the hub nodes [Gleicher et al. 2003] in the Motion Graph. We first categorize the initial and final postures of the interaction patches according to their relative distance, orientation and postures. The average poses of all the categories are computed and they become the standard poses. Then, we can concentrate on planning how to reach the standard poses. We use the locomotion engine for moving the characters to the desired locations when it is far away from the standard pose. The characters move towards the

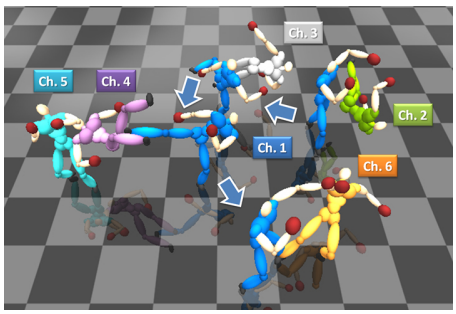


Figure 7: The scene that corresponds to the data flow shown in Figure 6 upper. The blue character (Ch.1) sequentially interacts with Ch.2, Ch.3, Ch.4 and Ch.6. This sequence of interactions is composed by temporal concatenation. Ch.4 falls over Ch.5. This interaction is produced by spatial concatenation.

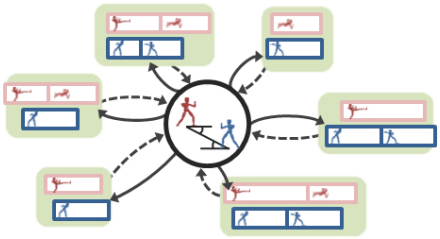


Figure 8: The standard pose (the circle at the center) acts as a hub to connect different interaction patches. The dotted lines indicate that the characters in the patches may need to adjust their locations and orientations for getting back to the standard pose.

nearest standard pose to start another interaction patch.

We define a distance function that evaluates the difference between the current pose (P_c) and each standard pose (P_s) as follows:

$$F(P_c,P_s)=(\frac{r_c-r_s}{r'})^2+(\frac{\theta_c^1-\theta_s^1}{\theta'})^2+(\frac{\theta_c^2-\theta_s^2}{\theta'})^2 \quad (2)$$

where r_c is the distance between the characters, θ_c^1 and θ_c^2 are the angles between the line connecting the two characters and the direction each character is facing, $r_s, \theta_s^1, \theta_s^2$ are the corresponding values in the standard pose. The constants r' and θ' are used to normalize the effects of distance and angle, and are set to $300cm$ and $180°$ respectively. The distance between the current status of the characters and each standard pose is calculated and the one with the smallest distance is selected:

$$argmin_{P_j} F(P_c,P_j) \quad (3)$$

where P_j is the j-th standard pose, and P_c is the current status of the two characters.

Once the target standard pose is selected, each character approaches the character it is to interact with by using the locomotion engine. When the characters are at the required relative distance and orientation, each character expands the game tree to find the action that brings its posture to that in the standard pose. Since (1) the connectivity of the action-level motion graph is high, and (2) the posture of each character in the standard pose is a commonly used posture, we can usually arrive at the target pose in one step. If the graph connectivity is low, and complex path planning is required for arriving at the standard pose, it is possible to apply dynamic programming to find the path in real-time.

As a result, even if there is no available interaction patch that can be immediately launched, the characters can move around and adjust their poses to start the next desirable interaction patch. As for timing, if one character arrives at the corresponding posture in the standard pose slightly earlier than the other character, we let the character wait there so that it is synchronized with its opponent before launching the next interaction patch.

4.4 Refining Motions

As motions during interactions require a lot of contacts and avoidance, we adjust the motions in order to preserve contacts or avoid penetration of the segments. We also need to refine the motions by reducing artifacts such as foot sliding due to the concatenation of motion clips. Such motion refinements are done at the final stage of the animation by traditional inverse kinematics and physically-based animation using the Open Dynamics Engine (ODE). When the segments collide, impulsive forces are added to the colliding segments to avoid penetration. Although the forces greatly change the posture, a PD controller is used to gradually pull the body back to the original motion [Zordan and Hodgins 2002]. If a response

motion is launched, the PD controller pulls the body towards the response motion [Zordan et al. 2005; Arikan et al. 2005].

5 Experimental Results

Using our method, we have simulated two types of scenes, which are generated by (1) only concatenating interaction patches, and (2) using the standard poses to let the characters continuously interact. The first group of scenes can be generated by the method explained in Section 4.2, and the second group of scenes further requires the techniques explained in Section 4.3. The set of *PatternList* used to generate the interaction patches are shown in Table 2.

Scenes generated by concatenating interaction patches: We simulated scenes where (1) a main character fights with many background characters (Figure 1a), (2) a group of people fall down over each other like dominos (Figure 1b, Figure 9, top), (3) an American football player holding the ball avoids the defenders and runs towards the goal (Figure 1c), and (4) a mouse runs into a crowd and the frightened people avoid it and bump onto each other (Figure 9, bottom, left). Although our system can automatically select all the actions for all the characters, usually the user prefers to give high level commands. Therefore, for scenes (1), (3) and (4), we have prepared an interface for the user to provide basic commands such as transition and rotation of the character, as well as field-specific commands such as punch, kick, and avoids. The commands correspond to $s_u(p)$ in Equation (1).

For scenes (2) and (4), the interactions of one character bumping into another or falling down onto another are designed by combining PD control and motion of being pushed away or falling down [Arikan et al. 2005; Zordan et al. 2005]. In scene (2), the interaction patches are automatically concatenated so that the area specified by a given bitmap on the floor is filled with characters falling to the ground. As the interactions between the characters are precomputed, even for large numbers of characters, we can obtain the results in real-time.

Scenes where characters are recycled: We simulated scenes where (1) two characters are continuously fighting (Figure 9, bottom right) and (2) a group of characters are passing luggage one after another to the characters next to them (Figure 1 (d)).

In the first example, after finishing an interaction patch, the characters either immediately enter another patch, or search for a standard pose, which leads them to a set of other patches. A fighting scene where the characters keep on attacking and defending can be generated. In the second example, each character continuously interacts with one of its neighbors. Different interaction patches are selected according to the size and the weight of the luggage. Each patch includes the motion of the first character standing, walking to receive the luggage, carrying and handing it to the second character, and going back to the original location. We define a set of standard poses which are suitable for passing and receiving luggage. Using these interaction patches and standard poses, we have generated a scene where a large number of characters pass luggage one after another to the next person.

Table 2: The *PatternList* used to compose the interaction patches (The actions of the second character are shown in bold font). Attack includes punch and kick, and defense includes dodge and avoid.

Scene	*PatternsList*
Fight (one-many)	{attack, **defense**, **attack**, fall}, {**attack**, fall}, {attack, **attack**, fall}, {arbitrary motion, **fall**, fall}
Fight (one-one)	{attack, **defense**}, {**attack**, fall}
Football	{run, **tackle**, avoid}
Mouse	{arbitrary motion, **avoid**, pushed away}, {arbitrary motion, **pushed away**, pushed away}, {run, **avoid**}, {arbitrary motion, **avoid**, fall}
Crowd falling	{arbitrary motion, **fall**, fall}
Luggage carry	{carry, **walk**, hand, **receive**, turn, **carry**}

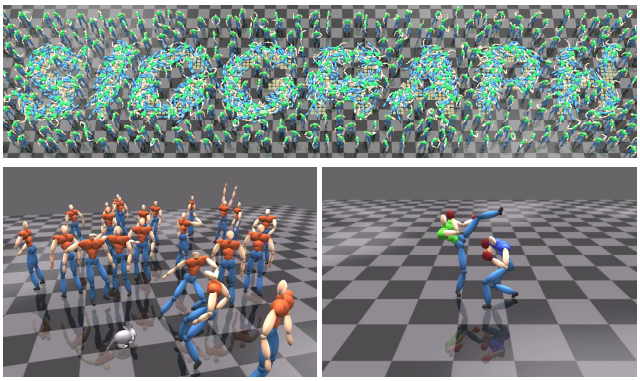


Figure 9: A large group of characters falling down onto each other in the floor bitmap area (top), a crowd in panic avoiding and bumping into one another (bottom left) and two characters continuously fighting in a stylized way (bottom right)

Table 3: The computational speed, number of actions and number of interaction patches of each experiment (Computational speed above 60 frame per second (fps) is real-time)

Scene	Speed (fps)	Actions	Patches
Fight (one-many)	87	162	157
Fight (one-one)	104	162	279
Football	194	217	21
Mouse	78	65	3716
Crowd falling	72	39	4091
Luggage carry	162	108	72

The computational speed and the number of actions and patches of each experiment are shown in Table 3. The computer used comes with a Pentium 4 Dual Core 3.0 GHz CPU and 2 GB of RAM. The reason for large numbers of interaction patches in the "Mouse" and "Crowd falling" demo is that we need to generate characters colliding from all directions for different orientations of the characters. Excluding the rendering, all the animation can be generated in real-time, once the instructions from the user are given. The readers are referred to the supplementary video for further details.

6 Summary and Discussions

In this paper, we have proposed a method to develop large-scale animations where characters have close interactions. The user can obtain stylized interactions between the characters by simply specifying the pattern of interactions. The interactions between the characters are saved by data structures called interaction patches. The interaction patches are spatio-temporally concatenated to compose large-scale scenes. Once the interaction patches are prepared, the process of composing the scene is fully automatic. At the same time, the users can control the scene using our control interface.

Our system requires far fewer samples than other optimization-based systems. For example, in [Shum et al. 2008], the number of samples produced is over 50,000. With this large number of samples, it is difficult to monitor the quality of the interactions. For our demos, fewer than 300 interaction patches are needed to create a stylized fighting scene. Previous methods of controlling characters are targeted for real-time applications such as computer games. In order to make the computer-controlled character strong, the controllability of the character must be high, which means the character

needs to be able to launch various kinds of movements, including subtly different steppings and attacks. This results in dense sampling of the state space. On the other hand, our objective is to create a stylized animation of characters interacting. The system does not need high controllability of the characters, but only needs to be able to satisfy the high level commands given by the animator. In addition to that, as our system first determines how the characters are going to interact, the characters have a lot of degrees of freedom to adjust their movements before and after the interactions. As a result, we can greatly reduce the number of interaction samples.

There are two possible extensions to enhance the controllability of the characters. The first method is to greatly increase the number of interaction patches and introduce a hierarchical structure to store the patches. In that case, according to the input by the animator, the corresponding cluster will be selected first, and then the best patch in the cluster will be selected subsequently. The second method is to introduce parametric techniques to deform and interpolate the existing patches. Using such a method, we will be able to produce a large number of variations from a small number of patches.

Our system can become an alternative to creating realistic interactions by using infinite horizon optimization methods such as reinforcement learning. In theory, it is possible to produce realistic interactions between characters if each of them select motions based on what benefits them the most. However, in practice, such smartness can make the scene less stylized as the characters will never conduct actions that do not benefit them. The characters become too careful and as a result, they will never launch risky movements that can make the interaction more attractive. On the other hand, the animators or the audience want to see energetic movements. It is much easier to produce such interactions by using our short-horizon method as the users can explicitly specify the pattern of interaction they want to see. Another advantage is that the computational cost is limited by the short depth of the game tree.

There are some limitations with our method. First of all, the process of specifying the pattern can cause problems if the actions by the characters are abstract and aimless as they are difficult to annotate. Our method is more suitable for actions which are easy to annotate. Secondly, we have limitations in generating scenes where multiple characters continuously interact. In the examples shown, the characters were allowed to adjust their movements without a time limit. If the time and locations of the interactions are strictly constrained, a global planner that can plan the sequence of all the characters at once will be required. Solving such a problem using discrete optimization is one of the future directions for this research.

Acknowledgement

We would like to thank the helpful comments of the reviewers. Our gratitude extends to Prof. Shigeo Morishima in Waseda University for arranging us several motion capture sessions. This project was partly supported by a funding from RIKEN, a CERG grant from the Research Grant Council of Hong Kong (Ref:CityU1149/05), and the Special Coordination Funds for Promoting Science and Technology from Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

Abe, Y., da Silva, M., and Popović, J. 2007. Multiobjective control with frictional contacts. *Proc of 2007 ACM SIGGRAPH / Eurographics Symp on Computer Animation*, 249–259.

Arikan, O., Forsyth, D. A., and O'Brien, J. F. 2005. Pushing people around. *Proc of 2005 ACM SIGGRAPH/Eurographics Symp on Computer Animation*, 59–66.

Gleicher, M., Shin, H. J., Kovar, L., and Jepsen, A. 2003. Snap together motion: Assembling run-time animation. *Proc of 2003 Symp on Interactive 3D Graphics*, 181 – 188.

Helbing, D., Farkas, I., and Vicsek, T. 2000. Simulating dynamical features of escape panic. *Nature 407*, 487–490.

Kwon, T., Lee, K. H., Lee, J., and Takahashi, S. 2008. Group motion editing. *ACM Trans on Graphics 27*, 3, 80:1–80:8.

Lau, M., and Kuffner, J. J. 2005. Behavior planning for character animation. *Proc of 2005 ACM SIGGRAPH/Eurographics Symp on Computer Animation*, 271–280.

Lee, J., and Lee, K. H. 2004. Precomputing avatar behavior from human motion data. *Proc of 2004 ACM SIGGRAPH/Eurographics Symp on Computer Animation*, 79–87.

Lee, K. H., Choi, M. G., and Lee, J. 2006. Motion patches: building blocks for virtual environments annotated with motion data. *ACM Trans on Graphics 25*, 3, 898–906.

Lee, K. H., Choi, M. G., Hong, Q., and Lee, J. 2007. Group behavior from video: A data-driven approach to crowd simulation. *Proc of 2007 ACM SIGGRAPH/Eurographics Symp on Computer Animation*, 109 – 118.

Liu, C. K., Hertzmann, A., and Popović, Z. 2006. Composition of complex optimal multi-character motions. *Prof of 2006 ACM SIGGRAPH / Eurographics Symp on Computer Animation*, 215–222.

Lo, W.-Y., and Zwicker, M. 2008. Real-time planning for parameterized human motion. *Prof of 2008 ACM SIGGRAPH / Eurographics Symp on Computer Animation*.

Park, S. I., Kwon, T., Shin, H. J., and Shin, S. Y. 2004. Analysis and synthesis of interactive two-character motions. *Technical Note, KAIST, CS/TR-2004-194*.

Reynolds, C. 1987. Flocks, herds, and schools: A distributed behavioral model. *Proc of SIGGRAPH 87 21*, 25–34.

Shum, H. P. H., Komura, T., and Yamazaki, S. 2007. Simulating competitive interactions using singly captured motions. *Proc of 2007 ACM Virtual Reality Software Technology*, 65–72.

Shum, H. P. H., Komura, T., and Yamazaki, S. 2008. Simulating interactions of avatars in high dimensional state space. *Proc of 2008 Symp on Interactive 3D Graphics*, 131–138.

Sung, M., Gleicher, M., and Chenney, S. 2004. Scalable behaviors for crowd simulation. *Comp Graph Forum 23*, 3, 519–528.

Treuille, A., Cooper, S., and Popović, Z. 2006. Continuum crowds. *ACM Trans on Graphics 25*, 3, 1160–1168.

Treuille, A., Lee, Y., and Popović, Z. 2007. Near-optimal character animation with continuous control. *ACM Trans on Graphics 26*, 3, 7:1–7:7.

Zordan, V. B., and Hodgins, J. K. 2002. Motion capture-driven simulations that hit and react. *Proc of 2002 ACM SIGGRAPH Symp on Computer Animation, 89–96*.

Zordan, V. B., Majkowska, A., Chiu, B., and Fast, M. 2005. Dynamic response for motion capture animation. *ACM Trans on Graphics 24*, 3, 697–701.

Zordan, V., Macchietto, A., Medina, J., Soriano, M., Wu, C., Metoyer, R., and Rose, R. 2007. Anticipation from example. *Proc of 2007 ACM Virtual Reality Software Technology, 81–84*.