Tiling Motion Patches

Kyunglyul Hyun, *Student Member, IEEE*, Manmyung Kim, Youngseok Hwang, and Jehee Lee, *Member, IEEE*

Abstract—Simulating multiple character interaction is challenging because character actions must be carefully coordinated to align their spatial locations and synchronized with each other. We present an algorithm to create a dense crowd of virtual characters interacting with each other. The interaction may involve physical contacts, such as hand shaking, hugging, and carrying a heavy object collaboratively. We address the problem by collecting *deformable motion patches*, each of which describes an episode of multiple interacting characters, and tiling them spatially and temporally. The tiling of motion patches generates a seamless simulation of virtual characters interacting with each other in a non-trivial manner. Our tiling algorithm uses a combination of stochastic sampling and deterministic search to address the discrete and continuous aspects of the tiling problem. Our tiling algorithm made it possible to automatically generate highly-complex animation of multiple interacting characters. We achieve the level of interaction complexity far beyond the current state-of-the-art that animation techniques could generate, in terms of the diversity of human behaviors and the spatial/temporal density of interpersonal interactions.

Index Terms—Three-Dimensional Graphics and Realism, Animation

1 Introduction

Simulating multiple characters interacting with each other is an emerging research topic in computer graphics. Modeling "interaction" between characters necessitates appropriate actions to occur in a carefully coordinated manner in both space and time. Precisely coordinating the spatial location, timing, and action choices of many interacting characters is challenging and often requires prohibitive computation and/or memory resource.

We are particularly interested in creating a dense crowd of virtual characters that interact with each other. The interactions may occur between characters in close vicinity and often involve physical contacts, such as hand shaking, pushing, and carrying a heavy object collaboratively. Each character may perform an arbitrary sequence of actions, which must be spatially aligned and temporally synchronized with other characters. Given a collection of action choices, each character can have exponentially many action sequences and only a small number of action sequences may satisfy stringent spatiotemporal constraints posed by interpersonal interactions. Finding such action sequences simultaneously for many interacting characters is extremely demanding.

We address the problem by collecting episodes of multiple characters. Each episode describes an interesting event featuring a small number of characters for a short period of time. Tiling episodes seamlessly across space and time would generate a crowd of densely interacting characters for an extended period of time. Each episode weaves a collection motion fragments to form a *motion patch* [1]. The motion patch has its *entries* and *exits*, which correspond to the beginning and end frames, respectively, of its motion fragments. Two motion patches can be stitched together by carefully aligning their entries and exits in both space and time. Our major challenge is tiling patches tightly such that the tiling does not have any dangling entries or exits, which cause sudden appearance and disappearance of characters in the crowd.

The tiling is a combinatorial problem of layering a collection of motion patches. It also involves continuous optimization of patch locations for packing patches densely while avoiding collisions. Feasible solutions are rare and exhaustive search is not feasible. Our tiling algorithm uses a combination of stochastic sampling and deterministic search to address the discrete and continuous aspects of the problem. Our motion patches are deformable to add flexibility in stitching patches and thus alleviate the difficulty of tiling. The deformable patch allows its motion fragments to warp smoothly while maintaining the coherence of interpersonal interaction captured in the patch.

Our algorithm makes it possible to automatically generate highly-complex animation of multiple interacting characters. Our algorithm takes raw motion data as input. The input motion data may capture a single subject or multiple subjects acting out in long clips. A collection of motion patches are identified semi-automatically from the input motion data. Perfect tiling of motion patches achieves the level of interaction complexity far beyond the current state-of-the-art that animation techniques could generate, in terms of the diversity of human behaviors and the spatial/temporal density of interpersonal interactions. The spatiotemporal nature of our tiling algorithm allows us to deal with both static and dynamic virtual environments. We can also control

• *K. Hyun, M. Kim, Y. Hwang, and J. Lee are with the School of Computer Science and Engineering, Seoul National University, Seoul, 151-744, Korea. Email: {kyunglyul, mmkim, youngseok, jehee}@mrl.snu.ac.kr*

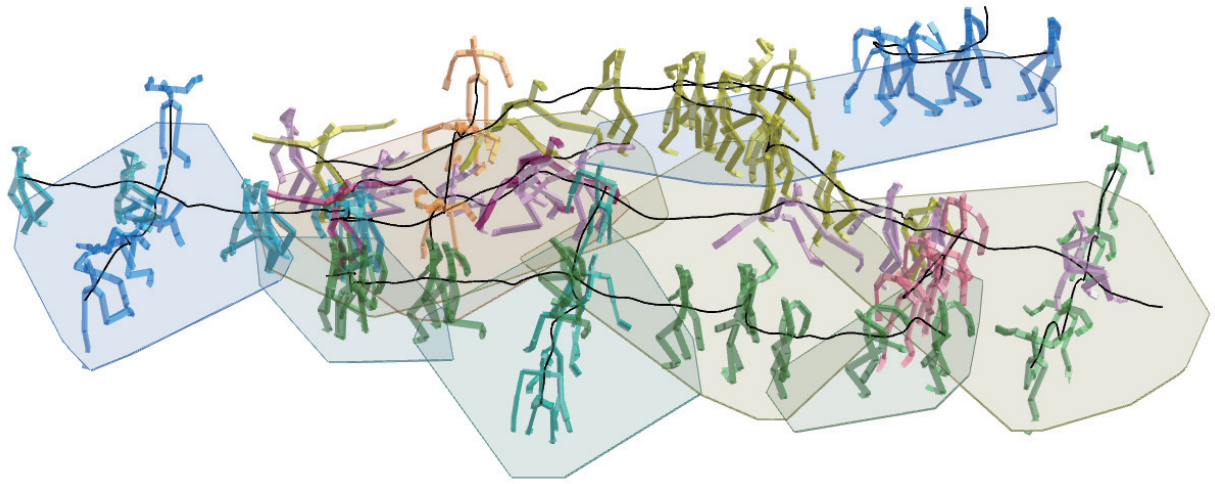


Fig. 1. Multiple characters are interacting with each other. A diversity of human behavior, such as jumping, crawling, pushing, bowing and slapping, are simulated in the automatically-generated, interpenetration-free animation.

the generation of animation by specifying when and which actions to occur at any specific location of the environment.

We further elaborate the tiling algorithm based on the reversible jump Markov Chain Monte Carlo (MCMC) framework [2] to allow the tiling to be updated incrementally according to the changes of the environment and user control. The reversible jump MCMC algorithm is a flexible method for solving discrete optimization problems, when the number of free parameters changes through the optimization process. In our problem, the dimension of the free parameter vector corresponds to the number of patch instances in the tiling. Reversible jumps in the MCMC algorithm enable the incremental insertion and deletion of motion patches.

2 Related Work

There exists a vast literature on crowd simulation in computer graphics. Crowd simulation refers to the process of simulating the movement of a number of pedestrians, which are equipped with action skills mostly for navigating in virtual environments and capability of replicating human collective behavior. Interaction behaviors, such as collision avoidance [3], [4], visual perception [5], and forming groups/formations [6] to name a few recent ones, have been explored. Multiple robot coordination pursues a similar goal with much emphasis on collision avoidance [7], [8]. Multiple robots are provided with start and goal configurations and search for collision-free paths simultaneously. It should be noted that our problem is different from crowd simulation and multi-agent path planning. We assume that interpersonal interactions would occur frequently and thus we search for a distribution of interactions rather than moving paths between start and goal configurations. Our characters are provided with a diversity of actions (beyond the scope of locomotion) to choose from. Every action should be precisely coordinated to meet alignment and synchronization posed by frequent interactions.

Research on multiple character interaction has recently emerged. Interactive manipulation techniques for editing multiple character motions has been studied with a varying level of control specification [9], [10], [11]. A lot of researchers have focused on interaction between two characters. Liu et al. [12] studied an optimization method to synthesize two character motions with physics-based objectives and constraints. Kwon et al. [13] simulated competitive interaction between two Taekwondo players using dynamic Bayesian networks. Komura and his colleagues addressed a similar problem using min-max search and state-space exploration [14]. Wampler et al. simulated two-player adversarial games based on game-theoretic behavior learning [15]. The state-exploration and learning-based approaches often suffer from the curse of dimensionality. Adding extra participants causes exponential increase in either computation time or memory storage usage. Thus, the techniques that worked well with two characters do not scale easily to deal with many characters.

In computer animation, metaphorical patches have been used to encapsulate either pre-computed or motion capture data. Tiling patches spatially and temporally generates animation of larger scales. Chenney [16] discussed the use of square tiles embedding flow patterns. *Motion patches* by Lee et al. [1] refer to a collection of building blocks of a virtual environment, which are annotated with motion data available on the blocks. Shum et al. [17] generalized the idea further to encapsulate short-term interactions between two characters into *interaction patches* and combined them to synthesize larger-scale interaction among many characters. Yersin et al. [18] tiled *crowd patches* to generate an arbitrarily large crowd of pedestrians in an on-the-fly manner. Our work is on the extension of these patch-based synthesis approaches. Given a collection of irregular patches, the

notorious problems of tiling are gaps (temporal discontinuity) and overlaps (interpenetration between characters). The temporal gaps between dangling entries and exits would cause characters to stay frozen over the gap or suddenly disappear and reappear at the boundaries. Our goal is tiling arbitrary irregular patches densely without gaps and overlaps.

The problem of packing irregular shaped pieces in a fixed area has been widely studied in geometric optimization and computational geometry [19] [20]. The shape packing idea has been exploited in various graphics applications. Kim and Pellacini [21] generated image mosaics by filling an arbitrarily-shaped container with image tiles of arbitrary shapes. Fu et al. [22] generated a set of quad-mesh to produce a tiled surface of computer-generated buildings. Merrell et al. [23] presented a method for automatic generation of residential building layouts. Packing and reconfiguring geometric shapes often pose complex optimization problems that involve both discrete changes to the geometric structure and continuous changes to the shape parameters. The MCMC sampling and its variants have been explored to address such complex optimization problems [24] [25]. Our approach were inspired by the research on geometric shape packing and we address further challenges that are unique in character animation. The challenges include the control of complex human behavior, handling multi-character interaction, identifying deformable motion patches as basic units of tiling, and the four dimensional tiling of deformable patches.

3 Deformable Motion Patches

Multiple characters interacting with each other create interesting episodes, which can serve as basic building blocks of complex multi-character scenes. Each episode can be modeled as a motion patch, which describes the actions of characters for a short period of time. Specifically, each motion patch includes a collection of motion fragments and their associated characters. Interaction between characters poses constraints between motion fragments. For example, handshaking indicates that the relative position and direction of two characters are constrained with each other while they are shaking their hands together. The motion patch may also include environment features and props that provide context to the characters' actions. The beginning and end frames of a motion fragment are called *entries* and *exits*, respectively, of the patch. We can stitch two motion patches if an exit of one patch matches an entry of the other. The character's position, direction, fullbody pose and timing should match within user-specified thresholds such that the character can make a smooth transition from one motion fragment to the other.

A *tiling* is a collection of motion patches that are stitched with each other. The tiling is *perfect* within a spatiotemporal region if it does not have any dangling entries or exits that do not have matching exits or entries in the tiling. More specifically,

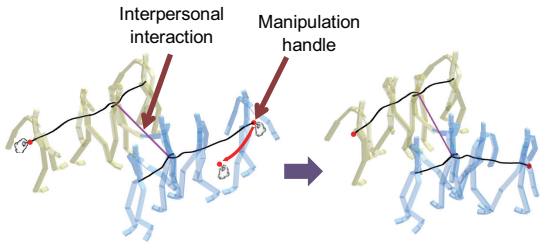


Fig. 2. The deformation of a motion patch, which shows two characters coming close, facing each other, handshaking, and walking away. The manipulation handles and interpersonal interaction are formulated as linear constraints. In this example, the interpersonal interaction is the relative body location and direction of two characters when their hands hold each other.

Patch Deformation Our motion patches are deformable to allow flexibility in tiling. The deformation is computed in two steps: Path deformation and fullbody motion refinement. At the first step, we consider a web of motion paths (see Figure 2). Each *motion path* is the trajectory of the skeleton root in a motion fragment. A local coordinate system is defined at every point on the motion path such that the position and orientation of the character's body can be represented with respect to the local coordinate system. A collection of motion paths are constrained to each other using linear interpersonal constraints. Given constraints, a motion path undergoes deformation as if it is an elastic string. The position/direction/time of entries and exits operates as if it is a manipulation handle, which can also be formulated as linear constraints. The manipulation handle is spatiotemporal in the sense that it specifies both body position/direction and the timing of the character's action. As a manipulation handle is dragged, a web of motion paths undergo deformation in an as-rigid-as-possible manner while maintaining a collection of linear constraints induced either from interpersonal interactions or user manipulation of motion paths. Specifically, we use multi-character motion editing by Kim et al. [10], which solves for the spatiotemporal deformation of multiple motion paths simultaneously subject to linear constraints based on Laplacian equations. The character's fullbody poses along motion paths are refined later to maintain hand-object contacts and rectify foot sliding artifacts [26].

Stitching Patches. We can stitch two deformable patches if there exist good matches at their corresponding entries and exits, through which characters can make smooth transitioning from one patch to the other. Let $\{(\mathbf{p}_i^A, t_i^A) \in \mathbb{R}^2 \times \mathbb{R} \mid i = 0, \cdots, N\}$ be a motion path in patch A, where $\mathbf{p}_i^A$ is the two-dimensional position of the character projected onto the ground surface and t_i^A is its timing. Note that the frames of motion data are usually sampled uniformly in time, though it is not necessary. The timing of motion or the interval of each individual frame may change as the motion data undergo deformation. The first frame $(\mathbf{p}_0^A, \theta_0^A)$ is an

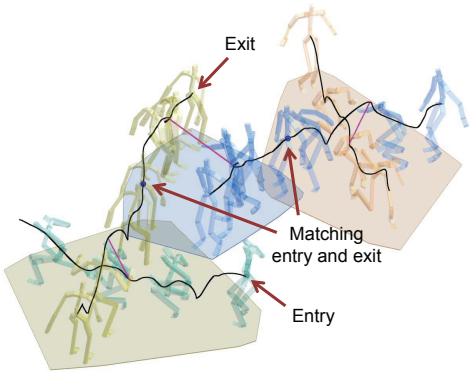


Fig. 3. Stitching deformable patches at their matching entries and exits. Each patch is depicted as a convex polygon that encloses its motion paths projected on the ground. The convex polygons are used only for visualizing the extent of motion patches.

entry of patch A and the last frame $(\mathbf{p}_N^A, \theta_N^A)$ is its exit. Assume that the dissimilarity between the configuration (the position, direction, timing, and fullbody pose) at an exit of patch A and the configuration at an entry of patch B is below user-provided thresholds. Concatenating the Laplacian equations of two patches into a single larger system of linear equations and adding three boundary constraints would deform two patches to stitch them seamlessly.

$$\mathbf{p}_N^A = \mathbf{p}_0^B \tag{1}$$

$$\mathbf{p}_N^A - \mathbf{p}_{N-1}^A = \mathbf{p}_1^B - \mathbf{p}_0^B, \tag{2}$$

$$t_N^A = t_0^B \tag{3}$$

which enforce smooth transiting in position, direction and timing, respectively. If there are more than one matching entry/exit pairs, we specify three transitioning constraints for each matching pair.

4 Patch Construction

Achieving a perfect tiling is challenging even though the flexibility of patches alleviates its difficulty to a certain degree. It is important that a collection of motion patches should be designed carefully for better connectivity. The motion patch is supposed to encapsulate an interesting episode that may entail complex spatiotemporal relationships among multiple characters. On the other hand, it is expected that the boundary (entries and exits) of the patch should be as simple as possible. The simplicity of the patch boundary facilitates the process of patch stitching.

Given a collection of raw multi-character motion data, we would like to construct a collection of motion patches to be used for tiling. It involves following steps (see Figure 4):

- Identify interesting episodes from raw motion data.
- Specify interpersonal and person-object constraints to preserve the spatiotemporal integrity of multi-character interaction under patch deformation.
- Determine the boundary of motion patches.

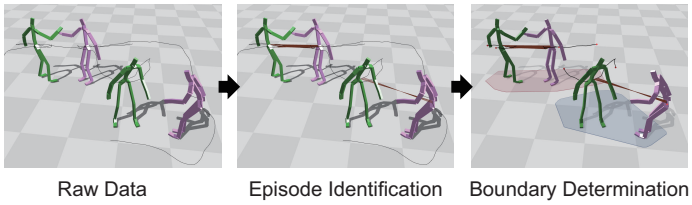


Fig. 4. The procedure of patch construction

Though there is no limitation on the size of motion patches, we usually design each patch to have less than five characters. If individual patches are too large, finding a perfect tiling could be difficult. Small patches including a single character are useful because they fit easily into narrow space. In practice, it is favorable to have a mixture of small, single-character patches and large, multi-characters ones for achieving good tilings.

4.1 Episode Detection

The episode refers to an event among multiple characters that is important or unusual. We use three criteria to detect such an event from raw motion data.

Contact. The physical contact between characters, such as handshaking, pushing, high-five, and punching, indicates the occurrence of interesting events.

Proximity. Even though there is no physical contact, it is noteworthy if two characters come close to each other and either of them reacts to the approach of the other. The presence of reaction is detected by measuring how much the character moves in a window of consecutive motion frames. The estimated variation of motion at frame i within window $[i-w, i+w]$ is:

$$V_i = \sum_{j=-w}^{w} \sum_{k=j+1}^{w} \|\mathbf{P}_{i+j} \ominus \mathbf{P}_{i+k}\|^2 \tag{4}$$

where $\mathbf{P}_{i+j} \ominus \mathbf{P}_{i+k}$ is the difference between two fullbody poses ignoring their horizontal translation and rotation about the vertical axis [1]. The value of V_i above a certain threshold implies there might be a noticeable action at frame i. The threshold has been chosen empirically by trial and error.

Synchronicity. Even though two characters do not come close to each other, it is also noteworthy if two characters perform interesting actions simultaneously and they are aware of each other's action. Assume that motion data include two characters A and B interacting with each other. Character A has its motion data with index i, and character B has its own data with index j. The i-th frame of A and the j-th frame of B are concurrent in the global, reference time. The synchronicity between two characters is:

$$S_{ij} = V_i^A \cdot V_j^B \cdot e^{-g(A,B)} \cdot e^{-g(B,A)}, \tag{5}$$

where $g(A, B)$ is the angle between the gaze direction of character A and the vector from the face of A to the upperbody of B. This term estimates the possibility of character A being aware of the action of B at the moment.

The detection of a noticeable event between two characters results in imposing interpersonal constraints between relevant motion data. The constraints enforce the relative position, direction, and their timing of actions remain unchanged even though their motion data undergo deformation. If we detect multiple events occurring nearby spatially and temporally, we merge them into a single, larger event, which may include interaction among more than two characters.

4.2 Boundary Determination

Once interesting events are detected, the next step is to determine their boundaries (entries/exits) to form a collection of tillable motion patches. There are several requirements. First, we would like to have the boundaries eventless. Any interaction/event occurring at boundaries would make the stitching process complicated. We want to have patches as compact as possible. Smaller patches tend to allow more flexibility in tiling and denser scenes to be created. Lastly, we want to have as little diversity of fullbody poses at boundaries as possible. Having a wide variety of fullbody poses at entries and exits would lead to difficulty of finding good matches in patch stitching. Note that we determine the boundaries of all patch collection simultaneously to satisfy the requirements.

Raw motion data include a multiplicity of fullbody poses at frames. Our algorithm for boundary determination begins by clustering fullbody poses into groups. Specifically, we use agglomerative hierarchical clustering such that the dissimilarity between any poses in a group is below a user-specified threshold. The threshold is chosen to allow smooth, seamless transitioning between motion fragments if the poses selected from a single group are at the matching entry and exit.

Let $\mathbf{G}_1, \cdots, \mathbf{G}_k$ be k largest pose groups sorted in descending order (typically, $k = 10$ in our experiments). We intend to choose a small number of groups that suggest candidate frames to be entries and exists. Then, the boundary may be determined to surround each event tightly at frames that belong to the groups. The number of groups used is directly related to the diversity of fullbody poses, so we would like to choose as fewer groups as possible. To do so, we consider all possible subsets, $\{\mathbf{G}_i | 1 \leq i \leq k\}$ of cardinality one, $\{(\mathbf{G}_i, \mathbf{G}_j) | 1 \leq i < j \leq k\}$ of cardinality two, and so on. We examine all possible subsets and choose the one that minimize the weighted sum of the cardinality of the subset and the total size (number of motion frames) of motion patches. The coefficient weighs the diversity against the compactness of motion patches.

5 Tiling Algorithm

The goal of tiling is packing patches in a user-specified spatiotemporal region without any dangling entries or exits while achieving the desired density of multi-character interactions and avoiding interpenetration between them. Specifically, our tiling algorithm is based on simulated annealing, which is known to be an effective sampling method for combinatorial optimization problems [27]. Though simulated annealing may have the potential to reach a global optimal solution, it takes a prohibitive computation time to remove dangling entries and exits completely. Our algorithm is a variant of simulated annealing, consisting of two phases. The first phase of the algorithm is intended to sample patches rapidly in the spatiotemporal region, while the second phase is a deterministic search and jittering procedure dedicated to removing dangling entries/exits.

Algorithm 1 Stochastic Sampling for tiling motion patches

$\mathbb{P}$: A tiling of motion patches
$R()$: A random number between 0 and 1

1: $T \leftarrow T_{\max}$
2: $i \leftarrow 0$
3: **while** $i < i_{\max}$ **do**
4: $\quad \mathbf{P}_{\text{new}} \leftarrow$ (a random patch instance)
5: $\quad \mathbb{P}' \leftarrow \mathbb{P} \cup \mathbf{P}_{\text{new}}$
6: $\quad \Delta E \leftarrow E_{\text{tiling}}(\mathbb{P}') - E_{\text{tiling}}(\mathbb{P})$
7: $\quad$ **if** $e^{-\Delta E/kT} > R()$ **then**
8: $\quad\quad$ **if** IsValid($\mathbb{P}'$) and NoCollision($\mathbb{P}'$) **then**
9: $\quad\quad\quad \mathbb{P} \leftarrow \mathbb{P}'$
10: $\quad\quad\quad i \leftarrow 0$
11: $\quad\quad$ **end if**
12: $\quad$ **end if**
13: $\quad T \leftarrow T - \Delta T$
14: $\quad i \leftarrow i + 1$
15: **end while**

5.1 Stochastic Sampling

Given a collection of motion patches, the first phase of our tiling algorithm is based on stochastic sampling of patches (see Algorithm 1). The position/direction/timing of patch instances are randomly chosen in a virtual environment and added to the tiling one by one (line 4–5). The sampling procedure minimizes an energy function

$$E_{\text{tiling}} = \frac{N_d}{N_c + N_d} + \alpha\left(\frac{1}{N_c + N_d}\right), \tag{6}$$

where N_c and N_d are the number of connected and dangling entries/exits, respectively. The first term penalizes the occurrence of dangling entries/exits, while the second term encourages rapid sampling of patches. As the number of patches in the tiling increases, it tends to have more dangling entries/exits. α is a constant that

balances between the density of patches and the number of dangling entries/exits. Each sample of a motion patch is accepted with certain probability based on simulated annealing (line 6–7). When the simulated temperature is high, the samples are easily accepted even though they do not improve the tiling energy much. As the temperature cools down, the system favors steady downhill traces to reach a stable solution. The Boltzman parameter k and the annealing schedule ($T_{\max}$ and ΔT) controls the rate of convergence. The algorithm terminates when the sampling procedure fails too many times (line 3).

Stitching, Deformation and Collision. As explained in the previous section, stitching two patches leads to a large system of linear equations that concatenates two systems of Laplacian equations. Adding a new patch to the tiling of patches involves all patches in the tiling and simply concatenating all linear systems into a single huge one is often infeasible (line 5). We assume that that the patches directly adjacent to the new one is deformable and all the others are pinned down. This assumption allows a practical trade-off between the computation time and the quality (in terms of flexibility and deformation) of tiling. The new patch is valid if it connects well to the tiling without incurring interpenetration or excessive deformation (line 8). We use a simple bounding box test on individual frames to check if there is interpenetration between two patches. We measure the degree of deformation on a per-frame basis. In our experiments, we allowed motion path deformation up to positional stretching of 3cm, bending of 7 degrees, and time shifting of 0.018 second per frame.

Algorithm 2 Deterministic search for removing danglings

$\mathbb{P}$: A tiling of motion patches
$\mathbb{D}$: A set of dangling entries and exits
$N(\mathbb{P})$: the number of dangling entries and exits in $\mathbb{P}$

```
1: while 𝔻 ≠ ∅ do
2:   for d ∈ 𝔻 do
3:     for (P_new that can resolve d) do
4:       ℙ' ← ℙ ∪ P_new
5:       if N(ℙ') ≤ N(ℙ) then
6:         i ← 0
7:         while i < i_max do
8:           if IsValid(ℙ') and NoCollision(ℙ') then
9:             ℙ ← ℙ'
10:            break
11:          end if
12:          ℙ' ← ℙ ∪ Jitter(P_new)
13:          i ← i + 1
14:        end while
15:      end if
16:      if d is not dangling then
17:        break
18:      end if
19:    end for
20:    if d is dangling then
21:      ℙ ← ℙ \ P(d)
22:    end if
23:  end for
24: end while
```

5.2 Deterministic Search

Even though the energy function used in the first phase tends to minimize the number of danglings, it usually converges very slowly (see Figure 6). The second phase of the algorithm is based on combinatorial exploration and randomized jittering in a continuous domain (see Algorithm 2). It examines every possible patch sequences from each dangling entry/exit until there remains no more dangling (line 2). Such an exhaustive search is infeasible in large free space. However, the patch tiling obtained from the first phase of the algorithm covers the target domain uniformly because of the nature of random sampling. The free space is fragmented into narrow spots, in which patch connection is strictly constrained. In the narrow, fragmented spaces, even exhaustive search is not computationally expensive any more.

For any dangling, the algorithm iterates over all possible patch choices that have an entry/exit matching the dangling (line 3). We first check if the number of danglings is increased by the new patch (line 5). Otherwise, we check further if the new patch incurs interpenetration or excessive deformation (line 8). If the new patch is not valid, we explore a continuous search domain by randomly jittering its spatial location, direction and timing (line 12). In our experiments, the jittering range is initially set as a circle of radius 0.6m in XZ-plane, an angle interval of -15 to 15 degree, and a time interval of -0.3 to 0.3 seconds. As the iteration continues, the jittering range expands gradually to explore a wider range. The maximum range is a circle of radius 1.0m, an angle interval of -25 to 25 degree, and a time interval of -0.5 to 0.5 seconds. If the dangling entry/exit has no patch to connect with it, the patch with the dangling entry/exit is removed from the tiling (line 20–21). The removal of the patch makes yet another dangling entries/exits. As the algorithm iterates, it will examine other possibilities that has not been explored before to eventually remove danglings completely (line 3). We keep track of sampling history so that patches removed in line 21 is not chosen again in line 3.

5.3 User Control

Though our tiling algorithm is capable of automatically generating a crowd of interacting characters, it can also allow the user to have direct and indirect control over the generation of animation. The user control may influence the motion of each character indirectly by specifying a variety of environment features or the ratio/preferences of actions. The user control can be more direct and immediate to specify which action to occur when and where

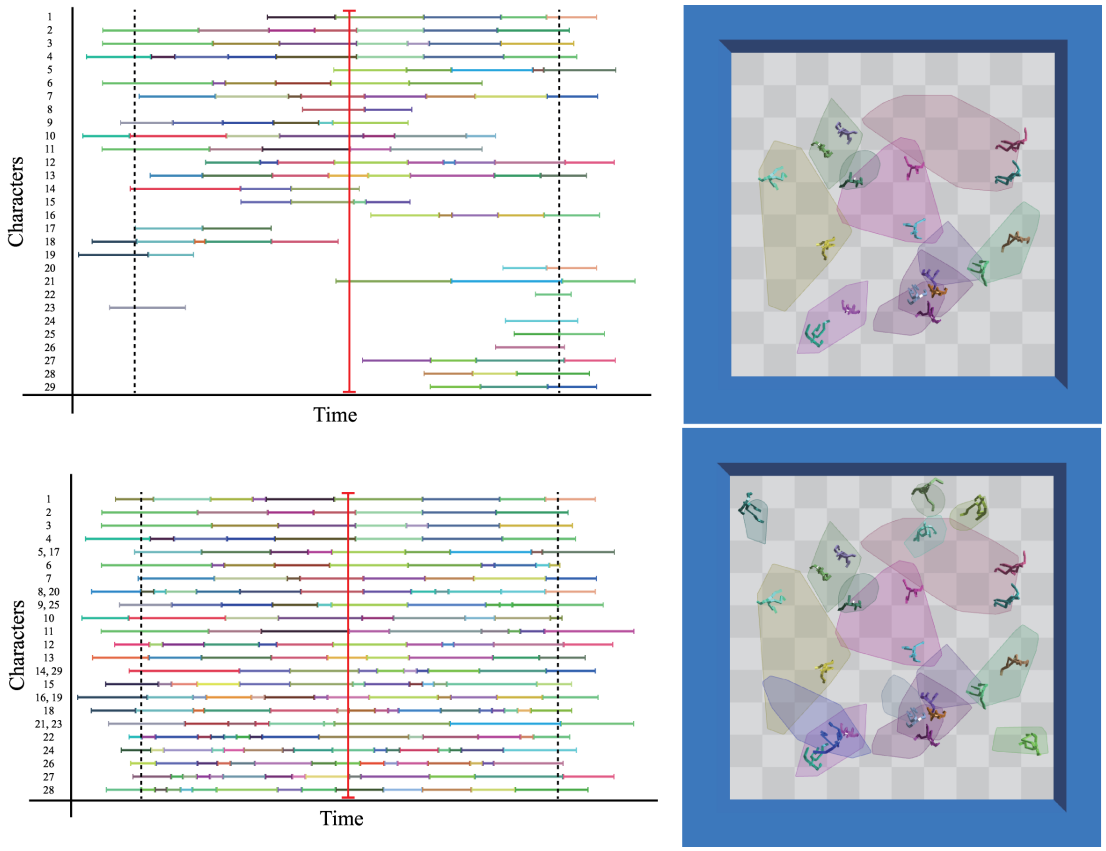


Fig. 5. A small example for illustrating the tiling procedure. The spatiotemporal volume consists of a square spatial region of 10 m $\times$ 10 m and a time interval of 600 seconds. The time interval is depicted as dashed lines. We cropped the animation at dashed lines. The screenshots on the right side were taken at the frame the red lines indicate. (Up) The first phase of our algorithm generated 67 patches featuring 29 characters. (Down) The second phase of the algorithm added new patches to fill in gaps. The final result included 197 patches featuring 23 characters. Note that the number of characters decreased, even though it has more patches. It is because the new patches filled in the gaps between the trajectories of characters 5 and 7, 8 and 20, 9 and 25, 16 and 29, and 21 and 23 to merge them.

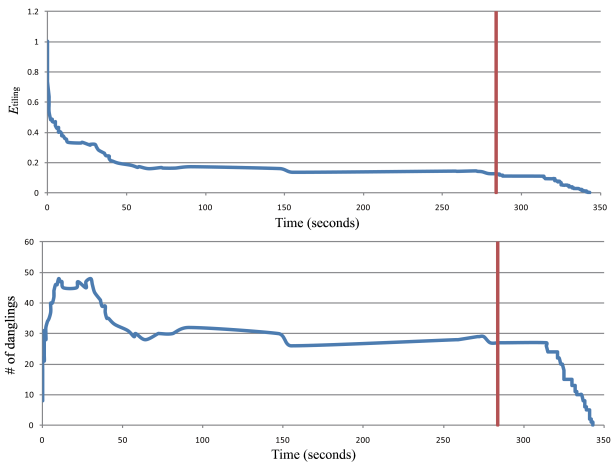


Fig. 6. The plots of the tiling energy and the number of dangling entries/exits with respect to the computation time. The red line shows the time when the second phase of the algorithm begins.

in the environment. We annotate each patch with labels, such as slow/fast and friendly/hostile, and allows the user to decide how frequently each class of actions to occur in the animation. More specifically, we define a preference function

$$F = e^{-f_1(x,y)/\sigma_1} \cdot e^{-f_2(\theta)/\sigma_2} \cdot e^{-f_3(t)/\sigma_3} \cdot e^{-f_4(l)/\sigma_4}, \quad (7)$$

where (x, y) is the position of a patch instance, θ is its orientation, t is the timing in the reference time, and l is its label. f_i controls the tendency of patch distribution and σ_i weighs the significance of the terms. For example, $f_1(x, y) = (x_d - x)^2 + (y_d - y)^2$ if we want to control the characters to move to their target location (x_d, y_d). If we do not have any preference over labels, then $f_4(l) = 0$ for any l. f_i can be defined in various ways to provide the user with flexibility and versatility in control specification. The preference function applies differently to the aforementioned algorithms. In the random sampling process (line 4, Algorithm 1), the normalized preference function is used as a probability distribution of sampling $\mathbf{P}_{\text{new}}$. In the deterministic search (line 3, Algorithm 2), the preference function prioritizes the candidates to de-

termine which patches to examine first.

Interactive Control. The second phase of the algorithm can be easily modified to deal with interactive control of a small number (less than a dozen) of characters. Two minor modifications are required. One is the way patch instances are sampled in time. While the original algorithm has a fixed time horizon, the interactive version of the algorithm advances the time horizon incrementally and patch instances are sampled continuously at runtime to fill up the horizon. The other is the way the tiling is deformed when a new patch instance is added. The original algorithm allows both the new patch and existing patches in the tiling to undergo deformation, while the interactive version allows only the new patch to deform. The rationale is that patch instances in the tiling is already in the past and should not change.

6 Local Update Algorithm

Given a perfect tiling of motion patches, we may want to update the tiling locally according to the changes of the environment and user control. This involves the insertion and deletion of motion patches incrementally from the tiling. The first phase of the algorithm described in the previous section only allows the insertion of patch instances. We employ a variant of the Reversible Jump Markov Chain Monte Carlo (MCMC) framework to design a local update algorithm. The simulated annealing is a classical MCMC method. The reversible jump MCMC provides better flexibility and generalization capability over simulated annealing.

6.1 Reversible Jump MCMC

Let $\mathcal{T}$ be the space spanned by given patches such that an element $x \in \mathcal{T}$ is a tiling of those patches. The dimension of x depends on the number of patch instances in the tiling. If we define probability distribution P on $\mathcal{T}$, we can sample random tilings from that distribution and it belongs to the class of MCMC sampling. Metropolis-Hastings algorithm is well-known technique for MCMC when direct sampling is difficult. The algorithm generates a sequence of random samples using probability distribution P and proposal density Q. Intuitively speaking, $P(x)$ explains how good the tiling is. The goodness is evaluated based on the target density, the number of danglings, and the number of collisions. Q elucidates our preference of which patches to sample where/when in the tiling. If we do not have any preference, Q is a uniform distribution (constant value). It takes new sample x' based on the current sample x and their proposal density. Then it either accepts or rejects the new sample with acceptance ratio

$$\alpha_{x\to x'} = \min\left\{1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)}\right\}.$$

If the new sample x' is rejected, the previous sample x is considered as next sample. Note that we only need the ratio of probabilities $P(x')/P(x)$, but exact probability $P(x)$ is not required. In the context of patch tiling, a new tiling x' is chosen by inserting a new patch instance to tiling x, removing a patch instance from tiling x, or jittering the location of a patch instance. Therefore, the dimension of x' may not match the dimension of x. The discrepancy in dimensionality could make the transition irreversible. The presence of irreversible transitions can be problematic, because the Markov chain cannot sample a certain portion of the space anymore and may lead to local optima.

Reversible jump MCMC is a variant of MCMC methods, which allows dimension jumping [2]. The dimension of tilings in $\mathcal{T}$ is not uniform, because the dimension of a tiling depends on the number of patch instances. The dimension matching function $f_{n,m}(x_m, u_{n,m})$ of reversible jump MCMC enables transitions between different dimensions. $u_{n,m}$ is a random variable which supplement generating a new state $x'_m \in \mathbb{R}^m$ from previous state $x_n \in \mathbb{R}^n$. The reversibility in dimension transition is guaranteed with acceptance ratio

$$\alpha_{x_n\to x'_m} = \min\left\{1, \frac{P(x'_m)Q(x_n|x'_m)}{P(x_n)Q(x'_m|x_n)}\left|\frac{\partial f_{n,m}(x_n,u_{n,m})}{\partial(x_n,u_{n,m})}\right|\right\}.$$

The proposal function $Q(x'_m|x_n) = J(n,m)Q(u_{n,m}|n,m)$ where $J(n,m)$ is the probability of jumping from dimension n to dimension m and $Q(u_{n,m}|n,m)$ is the proposal density of the random variable $u_{n,m}$. The Jacobian of the dimension matching function prevents irreversible jumps.

6.2 Tiling with MCMC

While incrementally updating a given tiling, we would like to minimize the energy function

$$E(\mathbb{P}) = w_e|\rho_o - \rho| + w_d N_d + w_c N_o^2,$$

where N_d is the number of danglings, N_o is the number of collisions, and ρ_o and ρ are the target and current density of tiles. w_e, w_d and w_c weighs three terms with respect to each other. In our experiments, weight values are 3, 0.02, and 0.01, respectively. The target probability is

$$P(\mathbb{P}) \propto e^{-E(\mathbb{P})}.$$

Note that we do not need to normalize the probability function, since only the ratio of probabilities $P(x')/P(x)$ is required in the sampling algorithm. We set jumping probability equally for insertion and removal.

$$J(n, n+1) = J(n, n-1) = \frac{1}{2}.$$

For $n \le m$, function $f_{n\to m}$ appends random variables to match the dimension of the source and target states such that $x_m = f_{n\to m}(x_n, u_{n,m}) = (x_n, u_{n,m})$ where $u_{n,m}$ is a uniform random variable that consists of the type, position, orientation, and timing of a new patch intance. We define the proposal density Q such that

$$Q(u_{n,m}) = U(\mathbf{C}) \times U(x,y) \times U(\theta) \times U(t),$$

Algorithm 3 Delayed Rejection Reversible Jump MCMC

$\mathbb{P}$: A tiling of motion patches
$R()$: A random number between 0 and 1

1: $accept \leftarrow 1$
2: $reject \leftarrow 0$
3: **while** $\frac{reject}{accept+reject} < threshold$ **do**
4: action ←INSERT or REMOVE
5: **if** action is INSERT **then**
6: $\mathbf{P}_{\text{new}} \leftarrow$ (a random patch instance)
7: $\mathbb{P}' \leftarrow \mathbb{P} \cup \mathbf{P}_{\text{new}}$
8: **else**
9: $\mathbf{P}_{\text{remove}} \leftarrow$ (a random patch in $\mathbb{P}$)
10: $\mathbb{P}' \leftarrow \mathbb{P} \setminus \mathbf{P}_{\text{remove}}$
11: **end if**
12: **if** $R() < \alpha_{\mathbb{P}\rightarrow\mathbb{P}'}$ **then**
13: $\mathbb{P} \leftarrow \mathbb{P}'$
14: $accept \leftarrow accept + 1$
15: continue from line 3
16: **end if**
17: $\mathbb{P}'_0 \leftarrow \mathbb{P}$
18: $j \leftarrow 1$
19: **for** $j \leq j_{\max}$ **do**
20: $d \leftarrow$ a random dangling entry or exit.
21: **for** $\mathbf{P}_{\text{new}}$ that can resolve d **do**
22: $\mathbb{P}'_j \leftarrow \mathbb{P}'_{j-1} \cup \mathbf{P}_{\text{new}}$
23: **if** not IsValid($\mathbb{P}'_j$) **then**
24: continue
25: **end if**
26: **end for**
27: **if** $R() < \alpha_{\mathbb{P}\rightarrow\mathbb{P}'_j}$ **then**
28: $\mathbb{P} \leftarrow \mathbb{P}'_j$
29: $accept \leftarrow accept + 1$
30: continue from line 3
31: **end if**
32: **end for**
33: $reject \leftarrow reject + 1$
34: **end while**

where $U(C), U(x, y), U(\theta), U(t)$ are the uniform distribution on types, locations, orientations, and timings of patch instances, respectively. Alternatively, we can use the preference function in Equation (7) to design the proposal density.

Ideally, the algorithm terminates if the tiling is perfect without any collision. In practice, achieving a perfect tiling based solely on MCMC sampling takes too much computation, if patches are packed tightly. Our MCMC sampling algorithm terminates if the rate of rejection is above a certain threshold. Then, we move on to the second phase of the algorithm presented in Section 5.2 to remove any residual danglings.

6.3 Delayed Rejection

Introducing a new patch into a tightly packed tiling often lead to a high rejection rate in MCMC sampling. Persistent rejection, often in particular parts of the tiling, is a computational bottleneck. The tiling is often locally stable and inserting/deleting a single patch instance cannot make a significant change to escape from the locally stable state. We employ a delayed rejection policy to alleviate persistent rejection [28]. On rejection of a proposal, a second subsequent attempt to move is made in hope that the subsequent move may resurrect the initial proposal. Specifically, when a jump from x_n to x'_m is rejected, we perform a repair step to generate a new sample x''_l. The composite move $x_n \rightarrow x'_m \rightarrow x''_l$ is either accepted or rejected with acceptance ratio

$$\alpha_{x_n \rightarrow x''_l} = \min\left\{1, \frac{P(x''_l)Q(x_n|x''_l)[1-\alpha_{x''_l \rightarrow x_*}]}{P(x_n)Q(x''_l|x_n)[1-\alpha_{x_n \rightarrow x'_m}]}\right\}$$

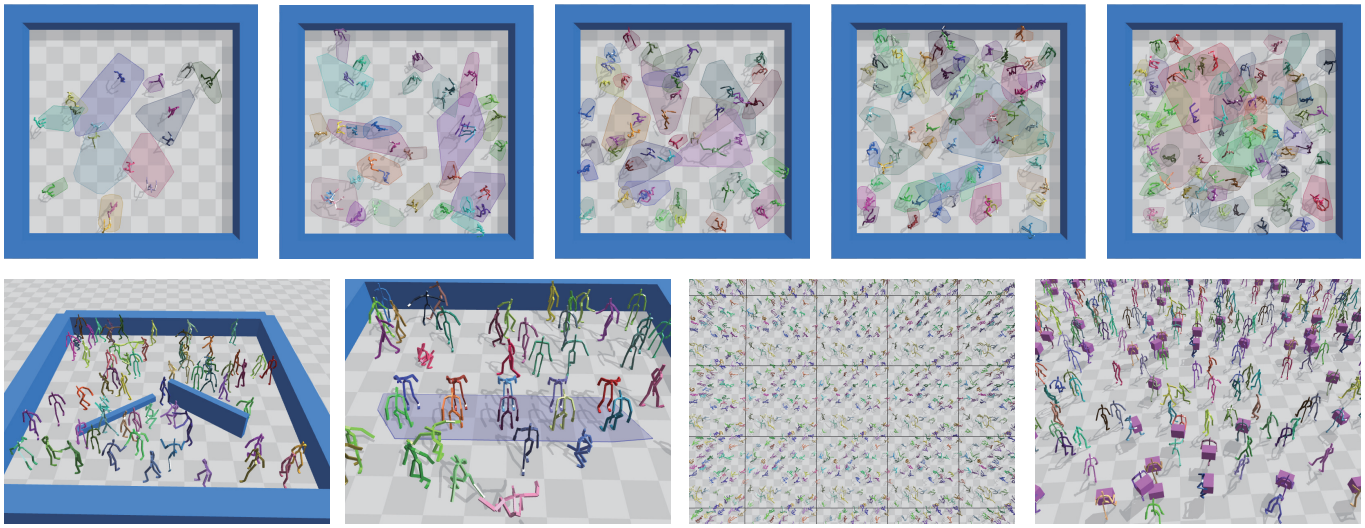
where x_* is x''_l with dimension matching such that $f_{n,m}(x_*, u_{n,m}) = x''_l$. We usually generate multiple secondary attempts for every proposal. The proposal is finally rejected if none of the composite moves are accepted. The reversible jump MCMC algorithm with delayed rejection is described in Algorithm 3.

7 Experimental Results

The timing data in this section was measured on a 2.93GH Intel Core i7 computer with 8Gbyte main memory and an ATI Radeon HD 4550 graphics accelerator. In our experiments, parameter α, k, $T_{\max}$ and ΔT are 0.2, 0.03, 200 and 1, respectively, for all examples. We do not need to tune parameters for each individual example.

Motion Patch Construction. We captured two subjects performing a variety of interactive actions, such as dancing, tagging, and bumping each other. Some actions involved a lot of physical contacts, while our subjects sometimes exhibited interaction without touching each other. We also designed some patches manually using our motion editing system we developed based on the idea of Kim et al. [10]. We constructed multi-character patches by putting together several pieces of single-person motion capture data and specifying interpersonal constraints. In our experiments, 43 kinds of motion patches are used and 27 of them are single-character patches. A big patch is used to demonstrate Keyframing example and 16 box-related patches are additionally used in Box-movers example.

Static Environments. Our first example was constructed in a static environment of 12m $\times$ 12m square. We generated an animation of 600 frames (20 seconds). In the first phase of the algorithm, we assigned preferences values to the patches such that larger (more characters, larger spatial extent, extended period of time) patches are more likely to be sampled. The second phase of the algorithm added mostly small, single-character patches that can easily squeeze into narrow spots (see Figure 5). The computation time increases pseudo-exponentially with the density of patches. Constructing dangling-free tiling with 173, 374, 490, 657, and 907 patches took 0.3,



Example	# of patches	# of patch types (# of unary patches)	# of characters	Size (m^2)	Time of Scene (seconds)	Computation time (minutes)	
						Phase1	Phase2
Static	173	43(27)	16	12 × 12	20	0.1	0.2
Static	374	43(27)	34	12 × 12	20	0.2	1.8
Static	490	43(27)	48	12 × 12	20	1	11
Static	657	43(27)	60	12 × 12	20	3	14
Static	907	43(27)	70	12 × 12	20	58	45
Dynamic	528	43(27)	66	12 × 12	20	53	44
Control	628	44(27)	55	12 × 12	23	17	14
Repeating	813	43(27)	55	10 × 10	27	83	43
Object	526	59(39)	36	10 × 10	27	1.5	5
Edit(*)	1463	43(27)	77	12 × 12	20	7	8

Fig. 7. Examples and Statistics. (*) denotes that the scene is generated using our incremental update algorithm.

2, 12, 17, and 103 minutes, respectively (see top row in Figure 7).

Dynamic Environment. Our spatiotemporal tiling algorithm is inherently capable of dealing with dynamically-changing environments. Whenever a new patch instance is sampled, the interpenetration with (either static or dynamically-moving) obstacles and other patch instances are detected and avoided. The environment may have sources and sinks where characters are newly created or disappear from the scene. The alignment of sources and sinks generates the flow of character's movements. The environment in the second example has two dynamically-moving obstacles (see left image, middle row in Figure 7). One of the obstacles is tall and the other is shorter. The characters can jump over the shorter obstacle, but cannot jump over the other. It took about 90 minutes to construct a tiling with 528 patches featuring 66 characters that playbacks for 20 seconds.

Chicken Hopping. We captured two subjects chicken-hopping, which is a play between human players. The player must stand straight, lift up one of his/her leg, hold the ankle with hands, and hop on the other leg. The players bump each other to make the opponent lose balance. We recorded 50 seconds of the play and our patch construction algorithm identified 6 two-player patches from the data. Many of the patches exhibit two players bumping and some patches shows one player dashes and the other evades. We manually picked 15 single-player patches that allow each individual character to navigate and steer. We made two demos using the data. The first demo shows two teams of players (red and blue) aligning at opposite sides, rushing towards each other, and bumping. Each team has 15 players. The player bumps the opponent, but avoids the fellow. The animation is 22 seconds long. The generation of the animation took 7 minutes. The second demo shows interactive control of small groups of characters. Each team has four players and the user selects target locations interactively to control them. Our interactive algorithm run on the average at the rate of 45 frames per second.

Keyframing. The user can directly control the position/direction/timing of specific patch instances in the tiling. To do so, the user selects several patch instances and arranges them in the spatiotemporal domain as he/she wants. We use the arrangement as the initial configuration of the tiling in Algorithm 1, which will add new patch instances around the user-specified instances to make them fully connected. The resultant tiling thus obtained will include the user-specified patches at desired spatiotemporal location. Specifically, we constructed a large patch with ten characters aligning in two

rows to bow to each other. We placed three instances of this large patch at 1.5, 11.5, and 21.5 seconds in the initial configuration of the tiling. The result shows the two-row formation is formed and dispersed repeatedly in complex multi-character animation (see the second image, middle row in Figure 7).

Spatially Repeating Tile. We can construct a seamless tile that repeats either spatially or temporal. Given a symmetric spatiotemporal region, the construction of repeating tiles allows patch instances to wrap around the symmetric region. Spatial repetition of tiles can cover an arbitrarily large spatial region seamlessly, while temporal repetition of tiles generates an infinitely long animation. Our 10m $\times$ 10m square tile repeats seamlessly on regular grids (see the third image, middle row in Figure 7). The repetition of square tiles is visible at large scale. It might be possible to construct aperiodic tilings, such as Wang tiles [29] and Penrose tiles [30], by sampling motion patches on triangular or rectangular regions with proper boundary conditions.

Box movers. In our box mover example, we have four patches that involve props. The character can pick up a box from a stack of boxes, carry it around, pass it to another character, put it down on the ground, and sit down on the box(see right image, middle row in Figure 7). The characters are precisely aligned and synchronized with each other when they pass a box from one character to the other.

Incremental Update. We generated a tiling for a static environment and updated the tiling to cope with a new obstacle at the middle of the environment (see Figure 8). Most parts of the initial tiling remains unchanged. Our local update algorithm efficiently removes interpenetrating patches and repairs the tiling to make it perfect without danglings. The generation of the initial tiling from scratch took 97 minutes. The local update is much faster, so it took only 15 minutes.

8 Discussion

We explored a patch-based approach to create a dense crowd of characters interacting with each other. The formulation was reduced to tiling spatiotemporal patches without temporal gaps and spatial overlaps. Our stochastic sampling algorithm demonstrated the power of patch-based approaches in creating very complex animation by tiling a small collection of precomputed motion patches.

Our new perspective allows complex interactions and physical contacts between multiple characters to be formulated as a sampling problem. Comparing to agent-based approaches, most of agent-based methods steer each individual character independently. Interpersonal interactions are formulated as heuristic rules or potential fields, but may not be enforced precisely. Our patch-based method would be reduced to an agent-based method if only single-character patches are used for tiling, because the string of single-character patches corresponds to the motion of each individual character. In that sense, the patch-based approach can be considered as a generalization of an agent-based approach.

There is no guarantee that the stochastic sampling converges while achieving all desired properties, such as the density of the crowd, collision avoidance, obeying user control, and coping with environment features. In practice, the convergence of the algorithm is closely related to the diversity and flexibility of motion patches. Including various single-character patches facilitates the convergence because it allows each individual character to be steered independently.

The collision detection and avoidance is a bottleneck of computation. More than half of the computation time is spent for bounding box tests and jittering-based collision avoidance. Collision avoidance can be more efficient if the penetration depth between two motion patches can be computed. Choi et al. [31] demonstrated an interactively-controllable character that is equipped with path planning capability and can navigate through highly-constrained environments in real-time. It was possible because all the obstacles had simple geometry and the interpenetration between the character and obstacles could be easily resolved. Unfortunately, we have no efficient algorithm to compute the penetration depth, which directly indicates the resolution of interpenetration, between four-dimensional spatiotemporal objects. Efficient computation of penetration depth would achieve significant performance gain for our tiling algorithm.

Patch-based motion synthesis has been explored by several research groups [1], [17], [18]. Specifically, we make two major improvements over the previous work. The first is the perfect tiling of motion patches. Shum et al. [17] has a lengthy discussion on why the perfect tiling is difficult and their observation motivated our work. We have found a solution to the problem that remain unsolved in the previous work. The two-phase algorithm and deformable patches played a key role in constructing the perfect tiling. The second improvement is the automatic algorithm for identifying motion patches from a collection of raw multi-character motion data. The overall contribution is a system that generates arbitrarily-large/long animations from a collection of raw motion data. The baseline algorithms are fully automatic. The animator may intervene to change and control the result by tuning parameters, defining control functions, and adding special features.

We presented two stochastic sampling algorithms. One allows only the insertion of patches to rapidly grow tilings. The other based on MCMC sampling allows both the insertion and deletion of patches to provide more flexibility and versatile. Theoretically, the latter algorithm subsumes the functionality of the former algorithm. In practice, there exists a trade-off between performance and flexibility. The former algorithm without patch deletion is usually faster than MCMC sampling when we want to generate a tiling from scratch. On the other hand, the flexibility of MCMC sampling allows us

Fig. 8. Incremental update. (Left) The tiling is initially generated in an empty space. (Middle and Right) Our reversible jump MCMC algorithm updates the tiling incrementally to introduce a new obstacle and enlarge the obstacle at the middle of the space.

to locally update existing tilings.

Patch-based motion synthesis has demonstrated its promise in controlling animated characters in large scale while coping with the complexity of environments, the diversity of human behavior, and the computation of long-term planning. The data-driven animation community have considered fragments of motion capture data as basic building blocks of human motion synthesis. We advocate motion patches as building blocks of large-scale motion synthesis.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (MSIP) (No.2013-003303 and No. 2007-0056094).

References

[1] K. H. Lee, M. G. Choi, and J. Lee, "Motion patches: building blocks for virtual environments annotated with motion data," *ACM Transactions on Graphics (SIGGRAPH 2006)*, vol. 26, no. 3, 2006.

[2] P. J. Green, "Reversible jump markov chain monte carlo computation and bayesian model determination," *Biometrika*, vol. 82, pp. 711–732, 1995.

[3] R. Narain, A. Golas, S. Curtis, and M. Lin, "Aggregate dynamics for dense crowd simulation," *ACM Transactions on Graphics (SIGGRAPH Asia 2009)*, vol. 28, no. 6, 2009.

[4] S. J. Guy, J. Chhugani, C. Kim, N. Satish, P. Dubey, M. Lin, and D. Manocha, "Clearpath: Highly parallel collision avoidance for multi-agent simulations," in *Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2009, pp. 177–187.

[5] J. Ondřej, J. Pettré, A.-H. Olivier, and S. Donikian, "A synthetic-vision based steering approach for crowd simulation," *ACM Transactions on Graphics (SIGGRAPH 2010)*, vol. 29, no. 4, 2010.

[6] E. Ju, M. G. Choi, M. Park, J. Lee, K. H. Lee, and S. Takahashi, "Morphable crowds," *ACM Transactions on Graphics (SIGGRAPH ASIA 2010)*, vol. 29, no. 6, 2010.

[7] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[8] R. Gayle, W. Moss, M. C. Lin, and D. Manocha, "Multi-robot coordination using generalized social potential fields," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2009, pp. 3695–3702.

[9] T. Kwon, K. H. Lee, J. Lee, and S. Takahashi, "Group motion editing," *ACM Transactions on Graphics (SIGGRAPH 2008)*, vol. 27, no. 3, 2008.

[10] M. Kim, K. L. Hyun, J. Kim, and J. Lee, "Synchronized multi-character motion editing," *ACM Transactions on Graphics (SIGGRAPH 2009)*, vol. 28, no. 3, 2009.

[11] E. S. L. Ho, T. Komura, and C.-L. Tai, "Spatial relationship preserving character motion adaptation," *ACM Transactions on Graphics (SIGGRAPH 2010)*, vol. 29, no. 4, 2010.

[12] C. K. Liu, A. Hertzmann, and Z. Popović, "Composition of complex optimal multi-character motions," in *Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2006, pp. 215–222.

[13] T. Kwon, Y.-S. Cho, S. I. Park, and S. Y. Shin, "Two-character motion analysis and synthesis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 3, pp. 707–720, 2008.

[14] H. Shum, T. Komura, and S. Yamazaki, "Simulating multiple character interactions with collaborative and adversarial goals," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 5, pp. 741 –752, may 2012.

[15] K. Wampler, E. Andersen, E. Herbst, Y. Lee, and Z. Popović, "Character animation in two-player adversarial games," *ACM Transactions on Graphics*, vol. 29, no. 3, 2010.

[16] S. Chenney, "Flow tiles," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004, pp. 233–242.

[17] H. P. H. Shum, T. Komura, M. Shiraishi, and S. Yamazaki, "Interaction patches for multi-character animation," *ACM Transactions on Graphics (SIGGRAPH ASIA 2008)*, vol. 27, no. 5, 2008.

[18] B. Yersin, J. Maïm, J. Pettré, and D. Thalmann, "Crowd patches: populating large-scale virtual environments for real-time applications," in *Proceedings of symposium on Interactive 3D graphics and games*, 2009, pp. 207–214.

[19] K. A. Dowsland and W. B. Dowsland, "Solution approaches to irregular nesting problems," *European Journal of Operational Research*, vol. 84, no. 3, pp. 506 – 521, 1995.

[20] V. J. Milenkovic, "Rotational polygon containment and minimum enclosure using only robust 2d constructions," *Computational Geometry*, vol. 13, no. 1, pp. 3–19, 1999.

[21] J. Kim and F. Pellacini, "Jigsaw image mosaics," *ACM Transactions on Graphics (SIGGRAPH 2002)*, vol. 21, no. 3, 2002.

[22] C.-W. Fu, C.-F. Lai, Y. He, and D. Cohen-Or, "K-set tilable surfaces," *ACM Transactions on Graphics (SIGGRAPH 2010)*, vol. 29, no. 4, 2010.

[23] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," *ACM Transactions on Graphics (SIGGRAPH ASIA 2010)*, vol. 29, no. 6, 2010.

[24] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun, "Metropolis procedural modeling," *ACM Trans. Graph.*, vol. 30, no. 2, pp. 11:1–11:14, Apr. 2011. [Online]. Available: http://doi.acm.org/10.1145/1944846.1944851

[25] Y.-T. Yeh, L. Yang, M. Watson, N. D. Goodman, and P. Hanrahan, "Synthesizing open worlds with constraints using locally annealed reversible jump mcmc," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 56:1–56:11, Jul. 2012. [Online]. Available: http://doi.acm.org/10.1145/2185520.2185552

[26] J. Lee and S. Y. Shin, "A hierarchical approach to interactive mo-

tion editing for human-like figures," in *Proceedings of SIGGRAPH 99*, 1999, pp. 39–48.
[27] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.
[28] P. J. Green and A. Mira, "Delayed rejection in reversible jump metropolis-hastings," *Biometrika*, vol. 88, pp. 1035–1053, 1999.
[29] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen, "Wang tiles for image and texture generation," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, no. 3, 2003.
[30] A. Glassner, "Penrose tiling," *Computer Graphics and Applications, IEEE*, vol. 18, no. 4, pp. 78 –86, 1998.
[31] M. G. Choi, M. Kim, K. L. Hyun, and J. Lee, "Deformable motion: Squeezing into cluttered environments," *Computer Graphics Forum (Eurographics 2011)*, vol. 30, no. 2, 2011.

Kyunglyul Hyun received the B.S. degree in Computer Science and Engineering from Seoul National University, Korea, in 2004. He is currently working toward the Ph.D. degree in the Department of Computer Science and Engineering, Seoul National University, Korea. His research interests include character animation and crowd simulation.

Manmyung Kim received the B.S. degree in Mechanical and Aerospace Engineering and the M.S. and Ph.D. degrees in Computer Science and Engineering from Seoul National University. He is currently a manager at Lineage Eternal Team, NCSOFT Corp. His research interests include synthesizing character animation and artificial intelligence.

Youngseok Hwang received the B.S. degree in Electrical and Computer Engineering and the M.S. degree in Computer Science and Engineering from Seoul National University. He is working in X3 Team, XLgames Corp. His research interests include computer graphics and animation.

Jehee Lee is a professor of Computer Science and Engineering at Seoul National University. He is leading the SNU Movement Research Laboratory. His research interests are in the areas of computer graphics, clinical gait analysis, and robotics. More specifically, he is interested in developing new ways of understanding, representing, and animating human movements. This involves full-body motion analysis and synthesis, biped control and simulation, animal locomotion, motion capture, motion planning, data-driven and physically based techniques, interactive avatar control, and crowd simulation. He served as a member of international program committees of top-tier conferences including ACM Siggraph, ACM Siggraph Asia, ACM/Eurographics Symposium on Computer Animation, and Pacific Graphics. He also served as a program co-chair of ACM/Eurographics Symposium on Computer Animation 2012.