Crowd Sculpting: A space-time sculpting method for populating virtual environments

Kevin Jordao, Julien Pettré, Marc Christie, Marie-Paule Cani

EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

Crowd Sculpting: A space-time sculpting method for populating virtual environments

K. Jordao[1], J. Pettré[1], M. Christie[2], and M.-P. Cani[3]

[1]Inria-Rennes, France
[2]University of Rennes 1, France
[3]Laboratoire Jean Kuntzmann, University of Grenoble & Inria, France

Abstract

We introduce "Crowd Sculpting": a method to interactively design populated environments by using intuitive deformation gestures to drive both the spatial coverage and the temporal sequencing of a crowd motion. Our approach assembles large environments from sets of spatial elements which contain inter-connectible, periodic crowd animations. Such a "Crowd Patches" approach allows us to avoid expensive and difficult-to-control simulations. It also overcomes the limitations of motion editing, that would result into animations delimited in space and time. Our novel methods allows the user to control the crowd patches layout in ways inspired by elastic shape sculpting: the user creates and tunes the desired populated environment through stretching, bending, cutting and merging gestures, applied either in space or time. Our examples demonstrate that our method allows the space-time editing of very large populations and results into endless animation, while offering real-time, intuitive control and maintaining animation quality.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism —Animation

1. Introduction

Large scale virtual environments are more and more present in the everyday applications we rely on (navigation, architecture, urban planning, games). Since these environments are mostly empty of virtual characters, there is a pressing demand to populate them in an easy and plausible way over long periods of time, to make them more lively, realistic and useful. However, the process of interactively designing a virtual population, even for medium-scale environments (such as a block of buildings), is not an easy task and requires addressing two main challenges: First, a crowd is an intrinsically complex system, containing characters that interact both at local and global scales, have individual or group behaviors and perform actions in close relation to their environment. The creation of realistic and complex crowds therefore requires the use of complex simulation or animation models for which the user needs to specify a large set of parameters through a series of, generally tedious, generate and test loops. Second, the required degree of control to easily and interactively populate such large environments requires the design of novel tools offering both spatial manipulation features (where to position the crowds, how to control their flow) and temporal manipulation features.

Unfortunately, techniques based on crowd simulation models only offer an indirect and generally global control on a crowd through the manipulation of agent parameters without immediate feedback [UCT04, PvdBC*11]. Another approach would consist in directly manipulating agents' trajectories [KLLT08, KHKL09], *i.e.* bending, stretching and shortening them to fit both user requirements and the topological constraints of the environment to populate. However, large deformations may lead to unrealistic results. The insight in this work is to replace the large deformations needed for sculpting a whole crowd by a number of local deformations and of local changes of animated content: using such changes will ensure that each piece of the crowd, while being able to continuously deform, only undergoes acceptably small deformations. Intuitively, by decomposing

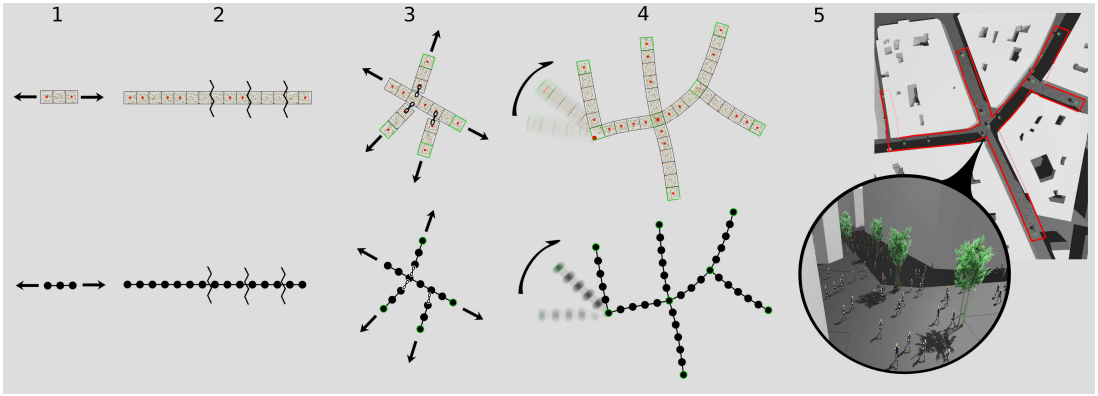

Figure 1: *Five steps illustrating the creation and interactive manipulation of Crowds Patches to populate a virtual environment by introducing spatio-temporal mutable elastic models. Pictures on the top line illustrate the manipulation of the patches while pictures on the bottom line illustrate the changes in the graph representing the Mutable Elastic Model.*

a large spatio-temporal crowd animation as a collection of pieces of animation (in a way similar to Finite Elements with deformable materials), each piece will follow an acceptable deformation while following the user's gesture, and be transformed to other content whenever the deformations become too large, as in plastic materials. This approach however leads to a number of challenges. Indeed, the overall spatial and temporal consistency of the crowd should be maintained, which requires continuity between crowd pieces (therefore between agent trajectories) both in space and in time. Moreover, sculpting should be allowed along the time line as well: users should be able to control the succession of trajectories used in each specific part of the scene. Lastly, they need to be provided with immediate visual feedback of the whole animated content, which is indeed the only way for them to achieve interactive design.

In this paper, we address these challenges, and propose a method for interactively sculpting crowd animations at large spatial and temporal scales. Unlike previous work, we do not play on simulation parameters (agents properties and goals) but directly manipulate trajectories. Unlike previous trajectory editing techniques, we manipulate picewise crowd animations to break deformation limitations both in space and time. This enable to precisely adapt them to the desired virtual environments to populate, and to finely control both spatial and temporal content (see Figure 1 for an illustrated example). In our framework, users design populated environments through simple interaction gestures such as stretching, shrinking or bending pieces of crowds, cutting and connecting pieces of crowds together and locally playing on motion variety over time. Meanwhile, local trajectory distortions are minimized and continuity of animations is ensured. Users immediately visualize the results as they interact with the crowds, enabling them to progressively populate and tune the crows behavior in the target environment. Technically, our solution consists in representing a crowd as an aggregation of spatially and temporally linked pieces of cycled animations, using the crowd patch principle proposed by Yersin *et. al.* [YMPT09]. In order to sculpt and deform the crowd patches (through stretching, bending, cutting tools) while maintaining the crowd structure, we extend the concept of Mutable Elastic Models (MEMs) [MWCS13], introduced for static shapes, to spatio-temporal crowd structures, leading to the new concept of 'mutable space-time models'. Our specific contributions are:

- a novel approach to interactively design complex animated crowds for virtual environments, with high level gestural control and immediate visual feedback. Note that the animations we generate can be endlessly looped in time, thanks to the crowd patches method we rely on.
- an extension of the mutable elastic models, introduced for shapes only, to enable the manipulation of structured geometries containing animations, while preserving spatial and temporal continuity between them.
- an extension of crowd patches that improves motion variety, thanks to our new temporal permutation mechanism. Given that patches capture periodic animations, we propose a technique to permute different version of a crowd patch over periods of time. Temporally extended motions inside a patch therefore present a more diversified content.

2. Related Work

Populating virtual environments requires animating a large number of agents. Such animation can be performed based on crowd simulation [TM07] techniques. Agent-based models provide animation trajectories, could they be based on sets of rules [Rey87], attractive and repulsive forces [HFV00], velocity-space analysis [PPD07, KO10, BGLM11] or synthetic vision [**?**]. Combined together with macroscopic techniques, they allow large and dense crowds [TCP06, NGCL09]. They are also coupled

with behavioral simulation to enrich content [ST07] or with environment annotations to provide meaningful animation [MHY*07,CIAC12]. Nevertheless, it is difficult to setup the simulation parameters in order to match a desired result; a lot of tuning is necessary both on the agent properties and on the agent goals. CrowdBrush enables interactively playing on some behavioural and aspects features [UCT04], but it is not possible to directly control the behavior of a crowd. Patil *et. al.* enable directing crowds based on navigation fields [PvdBC*11], but a simulation also depends on many other parameters and it remains difficult to predict the aspect of a crowd motion. In addition, the simulation of a large number of virtual agents remains computationally expensive. More importantly, in order to populate virtual environments in a realistic way, agents should display a sufficient degree of variety by performing a range of behaviors adapted to their local environment: adapting a simulation to a given environment is therefore difficult.

The second approach is based on the idea of reusing, combining or editing existing crowd animations. Simulation can be performed by solving interactions according to a set of examples [LCL07]. Models parameters can also be learned from real data [JCP*10]. And animations can be assembled to create nicely moving crowds [KHKL09, KHHL12]. Nevertheless, such approach requires large sets of data to provide enough variety. Furthermore, they do not provide users with a direct control to create a moving population. Closer to our concerns, a method was developed for generating new animations by interactively editing existing group motion data [KLLT08, KHKL09]. Granted that the initial motion quality is preserved, this brings the very important feature of being able to interactively deform animated content. However, this motion editing techniques cannot handle large special deformations and is only applicable to animations that are limited in time. Finally, Takahashi *et. al.* developed specific interpolation techniques to drive a formation through sequence of keyframes. Again, such method aims more at generating a finite animated sequence than to endlessly animate a moving crowd in a large and interactively explored environment.

The crowd patch technique [YMPT09] allows endless motion from a limited set of input animation. It partially overcomes the limitation of motion editing approaches by proposing means to assemble periodic pieces of crowd animations in order to create infinitely large, yet repetitive, animated crowds; nevertheless no interactive tools are offered to control their spatial or temporal layout. A challenge in the crowd patches technique is to compute animation trajectories which are periodic in time and which enforce specific limit conditions. Yersin showed that this can be done by starting from a set of constraints (a set of space-time way-points) and deducing trajectories [YMPT09] whereas Li showed how to extract patchable trajectories from existing motion data [LCS*12].

3. Overview

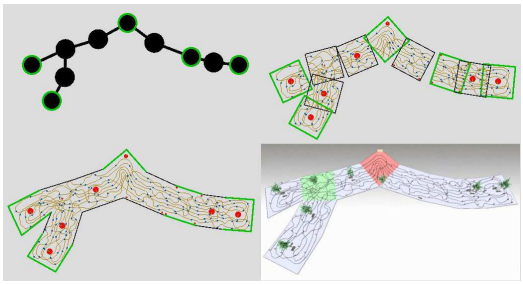

Figure 2: *A simple example to illustrate the principle of our approach.* ***Top****: A geometrical graph $\mathcal{G}$ captures the structure of crowd patches.* ***Middle****: Each vertex of $\mathcal{G}$ corresponds to a crowd patch.* ***Bottom****: Crowd patches are locally deformed to enable seamless animation continuity between patches.*

The principle of our approach is to represent an animated crowd as a graph-based structure in which nodes represent animated crowd patches [YMPT09], and edges represent the connected flows between the crowd patches. See Figure 2 for an overview: the top part represents the structure of the crowd as a graph, the middle part displays the non-deformed crowd patches corresponding to the graph nodes, and the bottom part represents the final deformed patches that shape the crowd animation and ensure continuity. From there, by proposing a novel extension of Mutable Elastic Models [MWCS13], user interactions applied to this representation can (i) impact the topology of the graph by inserting, removing or connecting nodes together thereby switching patches and changing the way patches are connected, (ii) impact the geometry of the patches (bending, stretching, shrinking thereby changing the way agents are moving inside a patch) or (iii) impact the temporal parameters inside the patches.

Lets first recall the principle and main properties of crowd patches in more details. A crowd patch is a delimited portion of pre-computed crowd animation. Animations can be of any kind: moving objects, pedestrians, vehicles, etc. A crowd patch is delimited both spatially (by a convex polygon) and temporally (by a period of time π). Animated objects smoothly move from patch to patch, thanks to continuity constraints enforced between adjacent patches: a pattern is defined on each edge of each patch to record the position and time at which objects traverse it; a mirror pattern (where an object going in becomes an object going out and vice versa) is given to the adjacent patch. All connected patches share the same period of time π, which allows smooth, endless animation to be produced by simply reusing the same trajectories over time.

In practice, crowd patches are organized into a bi-directional graph structure storing adjacency relationships.

Up to now, no feature was proposed to deform the individual patches or to change the topology or geometry of this graph. In addition, since time continuity was just obtained by replaying the patches content over time, no temporal structure was added. Our insight is to extend Mutable Elastic Models (MEMs) [MWCS13] to the crowd patches framework: MEMS are a unique solution for enabling large deformations of a complex structure while limiting the distortion of individual elements (here the geometry of crowd patches, and therefore of the trajectories). Whenever local deformation is too large, elements are either interactively swapped with more suitable ones, or some of them are inserted, removed, connected or dis-connected, as necessary, within the graph.

More precisely, MEMs were introduced for static shapes able to take a number of different *rest-states*, all preserving a meaningful structure. Let us take the example of a castle wall. Rest states for each section of the wall can either be "straight" or "corner". When the designer bends a straight wall beyond a certain point, one of the sections will turn into a corner, because this new rest position will best minimize the associated deformation energy. Interestingly, the structure of shapes is also captured in this work by a bi-directional graph representation $\mathcal{G}$, which deforms following the as-rigid-as-possible model [SA07]: deformation is computed so that the graph parts globally remain as close as possible to their rest states. The method therefore plays on the mapping between each graph node and rest states using a state space representation. Changes of the graph topology are also allowed, through node insertion and removals. Both changes of state and topology changes are based on a shape grammar, to ensure that only valid transformations are applied.

Although we inspire from the original idea, MEMs cannot be directly applied to our crowd patches framework. First, Milliez *etal.* did not solve for spatial continuity between the geometric elements they manipulate. In contrast, local deformations maintaining C_0 continuity at least is required in our case, in order to ensure smooth trajectories for virtual agents. Second, the elements of animation we manipulate require specific treatments to ensure proper temporal connections.

In the following section, we therefore present new, *mutable space-time models*, based on crowd patches. They require defining:

- a set of rest-state configurations, each rest-state representing a specific patch type and containing a patch instance (see Section 4.1);
- a state space which maps user interactions to changes in the topology of the graph structure (see Section 4.2);
- a mean to locally deform the geometry of crowd patches to follow the graph deformations (see Section 4.3);
- a visual way to manipulate animated content over the time line, in order to increase motion variety and control, making loops almost undetectable (see Section 4.4).

4. A mutable space-time model for sculpting crowds

The first step is to define a number of rest-states together with their corresponding patches and then the state space which describes the possibilities to swap rest-states or change the topological structure of the graph.

4.1. Patches Rest-states

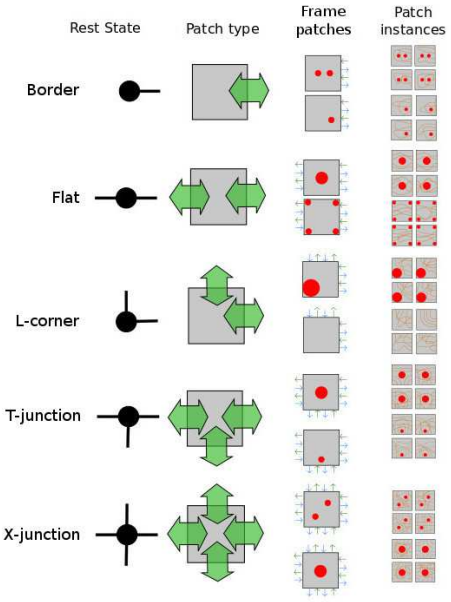

Figure 3: *Table of possible rest-states defined in our system (1st column), corresponding types of patches (2nd column), patches layout (3rd column) and examples of patches instances (4th column). The red dots represent obstacles in the patches.*

A rest-state is a predefined local configuration in the graph, *i.e.* a specific spatial arrangement of nodes and edges. Figure 3 illustrates the different rest-states available in our system, typically:

- a **dead-end** (1 edge only), $\chi^{(DE)}$,
- a **flat node** (2 aligned edges only), $\chi^{(F)}$,
- a **L-corner** (2 orthogonal edges), $\chi^{(L)}$,
- a **T-junction** (3 orthogonal edges), $\chi^{(T)}$
- or a **X-junction** (4 orthogonal edges), $\chi^{(X)}$.

Each rest-state is first associated with a patch type. The patch type typically defines the side on which agents flow between patches, yet without precisely defining densities of agents nor entry/exit patterns of agents. For the sake of simplicity, we restricted our patches to square-shaped regions. Each rest-state can therefore be connected to a maximum of 4 other patches. Each patch type is then associated with

a collection of *frame-patches* including only entry/exit patterns and obstacles, displayed in red in Figure 3. Each frame-patch is then associated to different patch instances which each contain different agent trajectories (all satisfying the entry/exit patterns and avoiding the obstacles), thereby providing a good degree of variability.

Frame-patches and Patch Instances: Frame-patches are delimited by patterns, which capture spacetime way-points for characters traversing patch boundaries. Frame-patches do not contain trajectories, but simply specify constraints. Patterns capture the flow going through each of the patch boundary. This means that we can easily deduce crowd patches instances from their type and frame-patch. We set empty patterns $p_\emptyset$ (without any way-point) where there is no flow allowed. We set a desired number of way-points for other patterns depending on the density of the patch. Combinations in the densities and in the patterns enable a first level of variety in the possible patches.

However, we must guarantee that the set of patches we use easily interconnects. Our solution is to create patches instances from a very limited set of patterns $\{p_1,...,p_n\}$ as well as their mirror patterns $\{p'_1,...,p'_n\}$. Indeed, we recall that two patches can connect if the spacetime way-points on their boundary match: one input point should corresponds to one output point in the adjacent patch. This condition is true for two mirrored patterns p_i and p'_i. By using a limited set of patterns, it is possible to pre-compute all the possible combinations of patterns to create different frame-patches, and then different patch instances. For example, an instance of a dead-end patch type patch is built with the following 4 patterns for each node: $\mathcal{P}_{dead\ end} \leftarrow \{p_i, p_\emptyset, p_\emptyset, p_\emptyset\}$. This patch can be connected for example to a flat-node type patch built with these 4 patterns: $\mathcal{P}_{flat} \leftarrow \{p'_i, p_\emptyset, p_j, p_\emptyset\}$.

Once patterns and obstacles are set in a frame-patch, different trajectories can be generated (the computation of such trajectories is not the purpose of this paper, see [YMPT09]). These combinations therefore enable a second level of variety in the possible patches.

Each time a new node N is inserted in the graph, all the patch instances of that node type for which all entry/exit patterns are compatible with its neighbors in the graph, are selected in a pool of possible candidates. A random process then chooses one among the possible candidates, and associates this patch instance with the node.

4.2. Patches State Space

We now describe how the graph deforms and evolves under a set of possible user actions, as well as how patches switch between different rest-states.

State Space: A state space is introduced to map each node in $\mathcal{G}$ with one rest-state. There is no ambiguity in this mapping when nodes are dead-ends, T or X-junctions, because the number of connected edges straightforwardly determines the corresponding rest-state. We however have an indetermination in the case of 2-edge nodes, that can either have a flat or a L-corner node rest-state. To solve this ambiguity, we define a state for nodes with two edges (rightmost green region in figure 4). Our state space is two dimensional. The state s_i of the node i is defined in the 2D state space as:

$$s_i = [l_i, \beta_i] \tag{1}$$

where β is the angle formed by the two edges connected to the node and l the length sum of the connected edges. We can now project the node into the state space. The state space is displayed in Figure 4. Depending on the position of the node in this space, we deduce its corresponding rest-state (color-coded areas in the state space). The figure also illustrates that, as detailed later, the user's actions will change the state of nodes, depending on their distance to the rest-states we defined. Distance and projection in the state space are computed according to the metric presented in [SA07] and based on the deformation energy $E(f)$:

$$E(f) = \sum_{p_i \in \mathcal{G}} dist(R_i(A_i), f(\chi_i)) \tag{2}$$

where $E(f)$ is the energy to minimize, p_i the graph nodes, and *dist* the distance between the configuration of the considered node $R_i(A_i)$ to its rest position $f(\chi_i)$. A_i is the set of possible rest-states $A_i = \{\chi_i^{(DE)}, \chi_i^{(F)}, \chi_i^{(L)}, \chi_i^{(T)}, \chi_i^{(X)}\}$. An example of mapping between the nodes of an example graph and the rest-states is displayed in Figure 5.

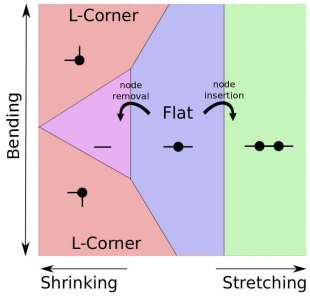

Figure 4: *Presentation of the state space. A colored region represents a state and boundaries represent transitions. User deformations move nodes away from their rest-state and inside this state space (the distance is measured using a specific metric integrating position and orientation information of edges). When reaching a boundary in the state space, the current rest-state is either switched to another one* (e.g. *Flat towards L-Corner) or leads to a change in the graph structure* (e.g. *node insertion or removal).*

Cutting and Merging: The user is able to cut or merge parts of the graph. These actions are direct and explicit actions on the graph $\mathcal{G}$. The user determines an edge to delete

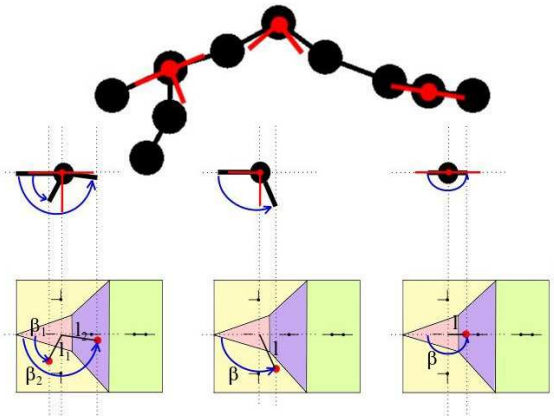

Figure 5: *Illustration of the deformation energy $E(f)$. Each node is associated a rest-state (closest one its state space). Here three rest-states are illustrated in red, each one associated to a state space in which the current configuration of the nodes (in black) are positioned. Distance between each node and its rest-state is then evaluated (using angles and edge lengths). Distances are then summed on all nodes to compute deformation energy.*

from the graph (cutting) or some new edges to add to the graph between two nodes (merging). In our application, the user performs cutting by selecting two adjacent patches and selecting the "cut" action. To merge two components of $\mathcal{G}$, the user moves one of the component (global translation / rotation) close to the other one, selects two close patches and select the "merge" action. An edge is created between the two corresponding nodes of $\mathcal{G}$.

Cutting and Merging actions directly result into changes of rest-states for the considered nodes. When cutting, the edge removal will mutate the connected nodes:

- X-junction into a T-junction: $\chi^X \rightarrow \chi^T$,
- T-junction into a flat node ("trunk" edge is cut): $\chi^T \rightarrow \chi^F$,
- T-junc. into a flat node ("branch" edge is cut): $\chi^T \rightarrow \chi^L$,
- Flat or L-corner node into a dead-end: $\chi^L, \chi^F \rightarrow \chi^{DE}$.

Note that we did not consider isolated nodes because this case is useless. And conversely, when merging, edge insertion will result into change in rest-state for the newly connected nodes: $\chi^{DE} \rightarrow \chi^L, \chi^F$, $\chi^L \rightarrow \chi^T$, $\chi^F \rightarrow \chi^T$. We choose whether χ^{DE} should turn into χ^L or χ^F by choosing the rest-state at closest distance from the current state of the considered node.

Node rest-state changes also result in a change of patches type (and consequently patch instances). The newly mutated node will connect to existing neighbors. Connection is made through patterns. These existing constraints define the correct patch-frame to be used. We then randomly select any instance of a patch with the corresponding patch-frame. Examples of cutting and merging operations are illustrated in Figure 1, steps 2 and 3. One can see flat nodes turning into dead-end nodes at step 2. In step 3, one flat node is successively merged with two other components: it turns into a T then into a X-junction node.

Stretching, shrinking and bending: Second, stretching, shrinking or bending operations can be performed on the graph as illustrated in Figure 1, step 4. The part to be deformed is selected by defining control nodes (represented as pins in our examples). The in-between structure of nodes is deformed in as-rigid-as possible way. As explained in [SA07], this is done computing the nodes configuration that minimize $E(f)$, Equation 2. This new configurations change the 2-edge nodes state.

Projection in the state-space (Figure 4 and 5) determines whether a swap in state (or mutation) is required. For example, when a flat node is stretched, its state is moved to the left of the state space: indeed, this move in the state space corresponds to a shortening of edge length. At some point, the state will go beyond the limit (between the pink and violet areas) where node removal is triggered. In the same way, when bending a flat node, the angle formed by node edges will increase or decrease: mutation to a L-corner χ^L rest-state is triggered (transition from the violet zone to the orange zone). A hysteresis at the frontier of different rest areas is implemented as suggested by Milliez to avoid disturbing oscillations between different rest-states.

As for cutting and merging, when a node mutates to a new rest-state or is added to $\mathcal{G}$, the matching patch-frame is selected and any instance corresponding to this frame is added to the animation.

4.3. Geometric deformations of patches

The elastic deformations controlled by the user cause some changes of distance and alignment between patches. As crowd patches contain some animated characters that move from patch to patch, un-alignment and distance changes result into discontinuities in the animation that need to be addressed (an aspect not tackled by [MWCS13]).

To ensure animation continuity, we propose to locally deform the patches geometry together with the animation to connect them seamlessly. Such a local deformation is illustrated in Figure 6. The results of a complex deformation is illustrated in Figure 2. Patches adjacency is easily deduced from $\mathcal{G}$. It allows us to map patch vertices of adjacent patches two-by-two. In the example of Figure 6, A_2 and B_1 are mapped, as well as A_3 and B_4. This allows to compute the new coordinates of locally deformed patches: they stand at the center of mapped vertices. For example, B'_1 is at the center of A_2 and B_1. Finally, all the internal trajectories as well as internal objects coordinates are deformed to follow the new shape of patches vertices. This local deformation is performed using a simple bilinear interpolation. Each trajectory

control point τ_{xy} is located at the same coordinates (α_x, α_y) expressed relatively to the moving axis $(\overrightarrow{B'_1B'_2}, \overrightarrow{B'_1B'_3})$. Given that deformations on the patches are limited (due to changes in states that arise) the bilinear interpolation only leads to small deformations on the trajectories.

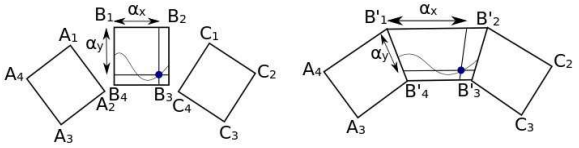

Figure 6: *Local deformation of patches. Vertices of two adjacent patches are interpolated. A bilinear interpolation of the internal animation trajectories is performed to fit the new patch position and shape.*

4.4. Interactive temporal editing

A crowd patch captures time-periodic animations. This enables endless replay of crowd motion. If a user carefully observes the same area for quite a long period of time, he may detect animation loop repetition. This is a drawback of the initial crowd patches we address here. In order to improve motion variety and lower the risk of animation loop detection, where needed (e.g., in the portion of the environment likely to be the most observed), we propose a mean to control the temporal variety. Our key-idea is to use different instances of patches at a given place instead of a unique one and play patch instances in sequence (patches permutation is performed at the end of the period).

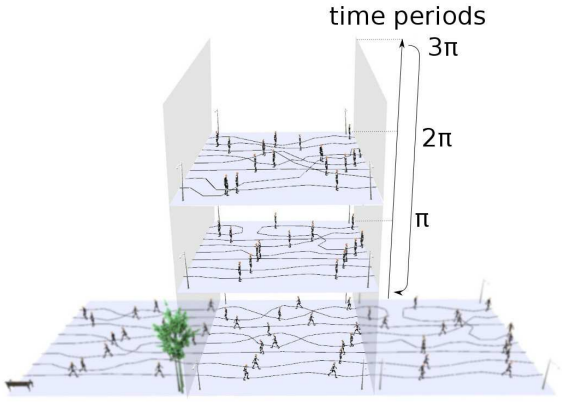

Figure 7: *Time permutation: three different instances of a patch built from the same patterns and initial state are used to animate this specific area. Patch instances of same period are looped in sequence. Given that initial states are the same, the patch instances are seamlessly permuted at the end of each period.*

In a given patch, trajectories are deduced from two constraints:

- a set of spacetime constraints at the limit of the patch: the patterns. Characters need to exit or enter patches through the spacetime waypoints as defined by patterns to enable smooth animations between two connected patches.
- the initial position of the characters in the patch at $t = 0$. Characters remaining inside the patch at the end of the period $(t = \pi)$ need to be located at the same positions than the ones where some characters were at the beginning $(t = 0)$. Note that different characters can occupy these same positions.

To enable time editing, we allow the possibility to enqueue different patch instances for each node of the graph and play them sequentially (all have the same period). These motions are computed using the constraints of the initial patch, to ensure continuity between neighbor patches. As a result motions inside a patch appear longer (for they are not repetitive) and much more different than a patch with shorter period. Finally, we compute several *versions*[†] of motions for a given patch instance to offer diversity when sculpting a crowd patch in time. Several versions of a patch are represented in Figure 8.

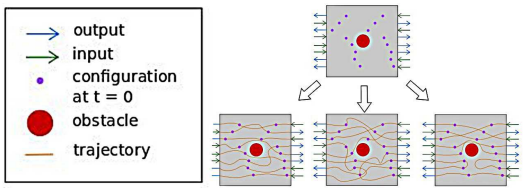

Figure 8: *Three different versions of a patch with similar patterns (*i.e. *blue and green arrows) and time boundary conditions (*i.e. *purple points).*

In the task of computing the animation trajectories of a given patch, the first step is to choose a mapping between each input and output spacetime waypoints as defined by the four patterns forming a patch. This mapping is a great help for the generation of motion variety. Indeed, to compute several variants of a given patch instance, we simply randomize the way we connect input and output points over all the ones defined by patterns. An example of insertion of 3 different versions of a same patch is displayed in Figure 7. Instead of one period of time, the motion will look different for at least 3 periods of time (however characters enter and leave the patch at the same time and position). Note that the selection order for versions of the patch can be randomized.

[†] Patches instance and patches versions are close notions. To help differientating them, in simple words, a patch instance is a set of spatiotemporal constraints that drive trajectories. A patch version is a patch with animation that satisfy those constraints. There can be several version of a same patch instance.

5. Results

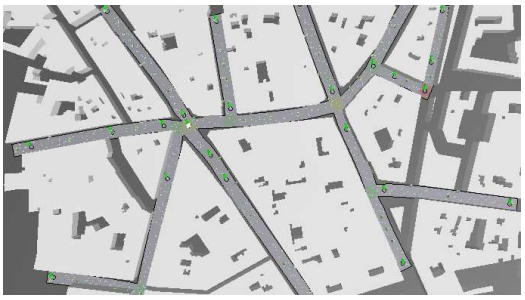

Figure 9: *Example of an animated city. Existing geometry is populated by crowd sculpting.*

We have implemented our crowd sculpting method with crowd patches structured in a mutable elastic model as a C++ library, using the boostgraph and the eigun libraries. On top of this library, two visualization components were developed for the sake of manipulation and visualization. First a simple 2D trajectory visualization component in which characters are displayed as moving points based on SFML. And second, a more complete visualization module based on the Unity Game Engine (see Figure 10). Note that in the 4.2 version we are using, Unity does not allow the display of more than 1.500 characters. Therefore our examples with more characters are only showed using our 2D visualization component.

As displayed in the companion video, interaction with the crowd is made using the mouse. The user selects the patches he wants to interact with by pinning them. The pinned patches are static or being moved by the mouse. The user then drags the selected patches to deform the structure. All the patches between to pinned and selected patches freely deform. As the patches are deformed all the trajectories are recomputed in real-time.

The crowd patches method is efficient, because computations are limited to data replay; complexity remains linear in the number of patches and characters. Together with deformation and the simple 2D visualization, the method is fast and able to deal with large environments. Based on the kind of examples we show in Figure 10, we are able to simulate the crowd and deform the structure of patches with smooth user experience (refreshing rate above 20 frames per seconds) up to 500 patches, which represents an average of 6000 characters (and given that our implementation is mono-thread, there is large room for improvement).

Figure 9 and 10 illustrate two results. In the first example, we deformed a structure of patches to match an existing environment. We reached the desired result in 15 minutes. These 15 minutes of Crowd Sculpting are almost entirely displayed in the companion video ($\times 10$ acceleration). In the second example, we created a crowd moving along the EG logo. We reached the desired result in 5 minutes after a dozen of deformations.

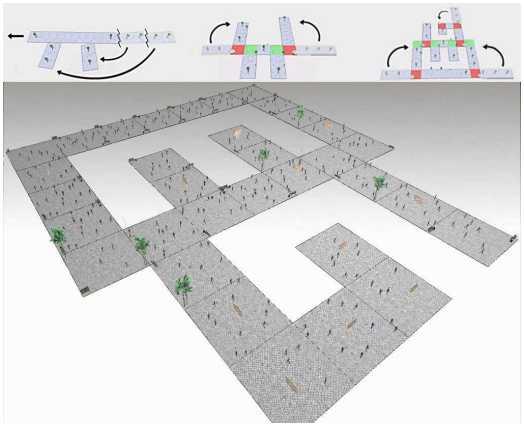

Figure 10: *Example of an animated crowd shaped as the Eurographics logo.*

6. Discussion and Future Work

The *crowd sculpting* method we presented is the first approach that makes the task of populating large virtual environments at the reach of any user. It offers a unique interaction experience with the complex animations in a crowd: indeed, designers can quickly manipulate a full crowd motion through a few sculpting gestures, enabling complex environments to be populated easily and quickly. Compared to crowd simulation approaches, we offer direct high-level control of trajectories, instead of indirect parameters. Compared to crowd motion editing techniques, we are not limited in terms of amount of deformation, animation duration or spatial coverage. Finally, compared to CrowdBrush, we do not edit some crowd simulation parameters but directly manipulate the coverage and shape of a crowd motion. We believe that it opens a new path for crowd motion design, by providing both new control paradigms and an interactive support for expressivity.

In the current version, our approach however displays several limitations, discussed next.

Extensions to multi-scale control: Our approach provides the users with a control over the spatial layout of patches and the local edit of motions in time. This control could be completed by larger or lower scales of control: at larger scales, it would be relevant to control global parameters, such as the evolution of global density in time (*e.g.* to reproduce daily cycle of a living city). At lower scales, it would be interesting to edit some individual trajectories, while automatically preserving the integrity of the patch (space and time constraints on the trajectories, typically entry and exit points).

Obviously, it can be difficult to control some local parameters in a single patch such as its density because densities of neighbor patches would be immediately impacted. Enabling accurate and local density control would require to extend

the types of patches approachby including *sinks* and *sources* in patches. Sinks and sources would be places where characters could appear or disappear (e.g., inside buildings). This simple extension would enable a finer control of densities, at the cost of linking sinks and sources to the scene geometry.

Animations trajectories: In this paper, we focused on a technique to interactively manipulate a given arrangement of patches. We did not consider the problem of computing animation trajectories inside patches; our examples were preapred using some existing method [YMPT09]. We illustrated our approach with patches generated from very few patterns (2 patterns + 2 mirror patterns). Although there is no limitation on the number and types of patch patterns, the fact that patch insertion can always be performed under stretching should always be guaranteed.

The specific problem of computing animation trajectories however need revisiting. First, the question of variety of patches needs addressing. Each animation trajectory is constrained by several spatiotemporal waypoints. Those waypoints enable time-periodicity or patches interconnections. How to get a *variety* of animation trajectories that satisfy those sets of constraints? Can we also consider secondary criteria such as: animation smoothness (inside and across patches), coherence of trajectories, improve animation richness and quality in a general way? Second, we should consider the use of perception studies to evaluate how detectible is the use of motion patches: what is the required variety to prevent spectators from detecting the use of patches? Is patches temporal permutation the most efficient solution to prevent animation looping? We have on-going works to address such question and we are currently exploring some formulation of the patch animation problem as an optimization problem to answer them. Perception studies could then follow. Finally, we also want to address the problem of computing patches for very specific situation such as for example traffic lights and

Trajectories lengthening & shortening: The different bending, stretching or shrinking operations on patches result in similar variations on the animation trajectories. Typically, this impacts the speed of walking characters (given that timing of entry and exit points in a patch are not changed). In our implementation, the maximum increase in speed is fixed to $\times 1.5$ the initial speed, a value above which node insertion is performed, bringing the speed back to $\times 0.66$ $\times 1.5$ the initial speed. The longer the strip of stretched patches, the smaller the proportion of speed change (for example 10 patches turned into 11 patch yield a $\times 1.10$ speed increase). However, such changes in speed may induce possible artifacts in animations (such as characters slowing down in large, straight, ways). An easy solution would be to extend the number patch types by including shorter or longer, rectangular shaped patches.

Also note that we use a linear interpolation to deform trajectories according to patch deformations. Although better methods for trajectory deformation could be thought of, we're facing a necessary trade-off between computational speed and quality of deformations. In our approach, we're more interested in maintaining an excellent level of responsiveness in the application (at the expense of deformation quality), and the user is probably not interested in having precise deformations in the interactive step (the result is more important than the interaction). Therefore a good solution would consist in keeping our current model for the stretching/compressing tasks, and switching to a better deformation technique once the shapes are at rest and the user is viewing the resulting animation.

Environments and animation content: Though we illustrate our approach with examples of pedestrian streets, our method is not limited to these environments. Green parks, public squares, public buildings, etc. are easily integrated by first, defining the set of rest states and second, mapping and computing the corresponding types of patches. For example, parks and squares would require a 2D stretching mode in addition to the 1D stretching used for pedestrian streets.

For populating very large environments, an improvement would be to provide a semi-automatic technique offering a balance between automated and interactive steps, *e.g.* automatically integrating the topological constraints of the environment, while offering all our other interactive features. For example, a designer would sculpt a crowd along the geometry of existing sidewalks and building walls, with the system automatically fitting the required width for the patch along these sidewalks and walls.

When populating an existing digital city (as shown in Figure 9), geometry need processing. Especially, we did not consider "small obstacles" in geometries (e.g., street lamps, trashes, etc.) . There are various ways to consider them. One is to remove them, as small objects can be incorporated in patches at the stage of their animation: characters trajectories will naturally avoid them. A second option is to deform animation trajectories when setting patches into the environment, but being able to solve collision is not guaranteed. Finally, it is possible to set patches so that they go around obstacles, but this may provoke complex patches layouts. We suggest combining the first and second solution, i.e., trying to incorporate small obstacles to patches when setting them, and to remove them if no solution is found to collisions.

Linear Deformation of patches: As detailed in Section 4.3, local deformations of patches are computed using a bilinear interpolation technique. While other techniques (such as group motion editing [KLLT08]) could be used to increase the quality of the resulting trajectories, the low computational cost of bilinear interpolations enables the simultaneous deformations on a very large number of patches.

7. Conclusion

In conclusion, we proposed the first method for interactively sculpting everlasting crowd motion at a large spatial scale. In addition to intuitive spatial control through deformation gestures, our method for temporal editing is one of the first solutions for providing interactive visual control of temporal content. Despite of the limitations we discussed, we believe that our work, enabling non-specialists to quickly shape lively environments, is a large step towards making authoring tools more widely available. Given that crowd-sourcing is an efficient solution for creating masses of 3D content, and that simple tools are already available for creating terrains and cities, our *crowd sculpting* approach could, in a near future, enable dedicated practitioners to populate online virtual worlds, with the activities of the virtual population evolving according to the time of the day, or even to the season.

Acknowledgment

This work is funded by the French National Research Agency ANR, projects CHROME and ISpace&Time. Part of this work was supported by the ERC Advanced grant "Expressive". The authors would like to thank Orianne Siret and Fabrice Lamarche for their help on preparing our demonstrations.

References

[BGLM11] BERG J., GUY S., LIN M., MANOCHA D.: Reciprocal n-body collision avoidance. In *Robotics Research*, Pradalier C., Siegwart R., Hirzinger G., (Eds.), vol. 70 of *Springer Tracts in Adv. Robotics*. Springer Berlin Heidelberg, 2011, pp. 3–19. 2

[CIAC12] CHARALAMBOUS P., ILIADOU H., APOSTOLOU C., CHRYSANTHOU Y.: Reconstruction of everyday life in 19th century nicosia. In *Proc. of the 4th int. conf. on Progress in Cultural Heritage Preservation* (2012), EuroMed'12, Springer-Verlag, pp. 568–577. 3

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating Dynamical Features of Escape Panic. *Nature 407*, 6803 (2000), 487–490. 2

[JCP*10] JU E., CHOI M. G., PARK M., LEE J., LEE K. H., TAKAHASHI S.: Morphable crowds. In *ACM SIGGRAPH Asia 2010 papers* (New York, NY, USA, 2010), SIGGRAPH ASIA '10, ACM, pp. 140:1–140:10. 3

[KHHL12] KIM M., HWANG Y., HYUN K., LEE J.: Tiling motion patches. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2012), SCA '12, Eurographics Association, pp. 117–126. 3

[KHKL09] KIM M., HYUN K., KIM J., LEE J.: Synchronized multi-character motion editing. In *ACM SIGGRAPH 2009 Papers* (2009), ACM, pp. 79:1–79:9. 1, 3

[KLLT08] KWON T., LEE K. H., LEE J., TAKAHASHI S.: Group motion editing. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 80:1–80:8. 1, 3, 9

[KO10] KARAMOUZAS I., OVERMARS M.: A velocity-based approach for simulating human collision avoidance. In *Intelligent Virtual Agents*, vol. 6356 of *Lecture Notes in Computer Science*. Springer, 2010, pp. 180–186. 2

[LCL07] LERNER A., CHRYSANTHOU Y., LISCHINSKI D.: Crowds by example. *Comput. Graph. Forum 26*, 3 (2007), 655–664. 3

[LCS*12] LI Y., CHRISTIE M., SIRET O., KULPA R., PETTRÉ J.: Cloning crowd motions. In *Eurographics/ ACM SIGGRAPH Symp. on Comp. Animation* (2012), Eurographics Association, pp. 201–210. 3

[MHY*07] MAÏM J., HAEGLER S., YERSIN B., MUELLER P., THALMANN D., VAN GOOL L.: Populating ancient pompeii with crowds of virtual romans. In *Proc. of the 8th Int. Conf. on Virtual Reality, Archaeology and Intelligent Cultural Heritage* (2007), VAST'07, pp. 109–116. 3

[MWCS13] MILLIEZ A., WAND M., CANI M.-P., SEIDEL H.-P.: Mutable elastic models for sculpting structured shapes. *Computer Graphics Forum 32*, 2pt1 (2013), 21–30. 2, 3, 4, 6

[NGCL09] NARAIN R., GOLAS A., CURTIS S., LIN M. C.: Aggregate dynamics for dense crowd simulation. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 122:1–122:8. 2

[PPD07] PARIS S., PETTRÉ J., DONIKIAN S.: Pedestrian reactive navigation for crowd simulation: a predictive approach. *Eurographics'07: Comp. Graph. Forum 26*, (3) (2007), 665–674. 2

[PvdBC*11] PATIL S., VAN DEN BERG J., CURTIS S., LIN M., MANOCHA D.: Directing crowd simulations using navigation fields. *Visualization and Computer Graphics, IEEE Transactions on 17*, 2 (2011), 244–254. 1, 3

[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proc. of the 14th annual Conf. on Comp. Graph. and Interactive Techniques* (New York, NY, USA, 1987), ACM, pp. 25–34. 2

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Eurographics Symposium on Geometry Processing* (Barcelona, Spain, 2007), Belyaev A., Garland M., (Eds.), Eurographics Association, pp. 109–116. 4, 5, 6

[ST07] SHAO W., TERZOPOULOS D.: Autonomous pedestrians. *Graph. Models 69*, 5-6 (Sept. 2007), 246–274. 3

[TCP06] TREUILLE A., COOPER S., POPOVIĆ Z.: Continuum crowds. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, ACM, pp. 1160–1168. 2

[TM07] THALMANN D. T., MUSSE S. R.: *Crowd Simulation*. British Library, 2007. 2

[UCT04] ULICNY B., CIECHOMSKI P. D. H., THALMANN D.: Crowdbrush: Interactive authoring of real-time crowd scenes. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation* (2004), SCA '04, Eurographics Association, pp. 243–252. 1, 3

[YMPT09] YERSIN B., MAÏM J., PETTRÉ J., THALMANN D.: Crowd patches: populating large-scale virtual environments for real-time applications. In *Proc. of the 2009 symp. on Interactive 3D graphics and games* (2009), I3D '09, ACM, pp. 207–214. 2, 3, 5, 9