Crowd Patches: Populating Large-Scale Virtual Environments for Real-Time Applications

Barbara Yersin *
Jonathan Maïm
VRlab-EPFL, Lausanne, Switzerland

Julien Pettré †
Bunraku team, INRIA
Rennes, France

Daniel Thalmann
VRlab-EPFL
Lausanne, Switzerland

Figure 1: *Environments successfully populated with crowd patches. (Left) A procedurally computed pedestrian street, where patches are generated at run-time. (Middle-left) The same image revealing the patch borders and their trajectories. (Middle-right) A pre-defined city environment where patches are computed offline. (Right) The same image with apparent patch borders and trajectories.*

Abstract

Populating virtual environments (VEs) with large crowds is a subject that has been tackled for several years. Solutions have been proposed to offer realistic trajectories as well as interactivity, but limitations remain on the environment dimensions with respect to population density. In this paper, we extend the concept of motion patches [Lee et al. 2006] to densely populate large environments. We build a population from a set of blocks containing a pre-computed local crowd simulation. Each block is called a *crowd patch*. We address the problem of computing patches, assembling them to create VEs, and controlling their content to answer designers' needs. Our major contribution is to provide a drastic lowering of computation needs for simulating a virtual crowd at run-time. We can thus handle dense populations in large-scale environments with performances never reached so far. Our results illustrate the real-time population of a potentially infinite city with realistic and varied crowds interacting with each other and their environment. We discuss the advantages and drawbacks of the proposed solution, and its possible improvements in the future.

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

Keywords: interactive crowd, virtual environment population

1 Introduction

For a long period of time, VEs have been sparsely inhabited by virtual people. More recently however, they have benefited from computers' power increase, especially in the field of graphics capabilities. As a result, it is now possible to interactively model densely inhabited worlds. Efficient crowd rendering engines, able to display thousands of virtual humans in real-time, have been developed: a typical approach is to store pre-computed images of individuals, and reuse them on the fly. In addition, individuals also have to act and navigate in their environment. This is the role of *simulation*, which requires real-time methods in the case of VEs. Crowd rendering engines and simulators are thus combined to create interactive inhabited VEs. However, crowd simulation remains computationally expensive, even when virtual humans' behavior is limited to navigation. This is especially due to the costly avoidance of collisions. One possibility is to pre-compute, store and replay a simulation. Nevertheless, the content is then fixed, and limited in duration and size for obvious memory storage reasons. Moreover, VEs can reach huge dimensions, like entire cities or even up to the entire Earth, making the pre-computation of an animation for a corresponding virtual population unthinkable. We address in this paper the problem of densely populating any virtual environment without any assumption or explicit limitation on its dimensions, the possible duration of its exploration, or its population size.

Motivations. Our motivations in this work are threefold. First, we want to consider environments of any dimensions, from small public areas to entire cities, and potentially more. The problem raised by these environments is the underlying need for large-scale virtual populations. Second, we want to handle environments procedurally generated online and in real-time. It is difficult for current techniques to populate such environments, because their geometry cannot be pre-processed - a typically compulsory stage in crowd simulations. Third, we want to lower the relative cost of motion synthesis and navigation simulation, with respect to the whole VE resource needs, with no visible impact on the resulting motion quality.

Contributions. Our contribution consists of four points. First, we propose a technique that pre-computes simulation tasks, stores the resulting motion trajectories, and reuses them on the fly. Obviously, such pre-computations allow us to drastically decrease the online resources needed to animate virtual populations. We break classical crowd simulation limitations on the environment dimensions: instead of pre-computing a global simulation dedicated to the whole environment, we independently pre-compute the simulation of small areas, called crowd patches. To create virtual populations,

*e-mail: firstname.lastname@epfl.ch
†julien.pettre@inria.fr

the crowd patches are interconnected to infinity from the spectator's point of view. We also break limitations on the usual durations of pre-computed motions. By adapting our local simulation technique, we provide periodic trajectories that can be replayed seamlessly and endlessly in loops over time. As a second contribution, we propose a user-guided design technique to control the environment content in an easy and efficient way. Our technique is based on a set of *patch templates*, having specific constraints on the patch content, *e.g.*, the type of obstacles in the patch, the human trajectories there, etc. A large variety of different patches can be generated out of a same template, and then be assembled according to designers' directives. Our third contribution is the ability to populate any environment, whether it is already provided, or procedurally modeled in real time: patches can be pre-computed to populate the empty areas of an existing virtual environment, or generated online with the scene model. In the latter case, some of the patches will also contain large obstacles such as the buildings of a virtual city. Finally, our fourth contribution is to offer major improvements on the difficult trade-off between simulation quality, memory consumption, and performances. Concerning the quality of the crowd simulation, since it is pre-computed, fine-grained offline techniques for motion synthesis and simulation can be used. Secondly, since the resulting trajectories are periodic, the memory consumption is no longer directly related to motion duration: potentially infinite motions are achievable from a limited set of patches. Finally, run-time operations are limited to motion replay, resulting in high performances.

Overview This paper introduces the crowd patches approach, whose objective is to populate an environment from a set of blocks containing pre-computed motions for virtual humans. By constraining motion trajectories both in space and time at the limits of each block, we can connect them to build large environments with numerous humans. In Section 2, we first present previous work on real-time crowd simulation and rendering. In Section 3.1, we introduce our elementary block structure, the *crowd patch*, that may contain static or dynamic objects. Virtual humans composing our population are able to move from patch to patch. As a result, trajectory continuity must be temporally and geometrically ensured between patches. This is the role of *patterns*, introduced in Section 3.2. We show how to compute patches from patterns in Section 4: the most difficult point is to compute periodic motions while respecting continuity conditions on patch borders. The content of patches must fit the designers' requirements on the VE's content; we thus introduce *patch templates* in Section 5.1, with key-ideas for patching environments. Templates allow us to define the expected content of patches with respect to their location in the environment. They also provide control on the population motion and its distribution in different areas. We demonstrate our approach in Section 6 with two examples. Finally, we conclude with a discussion on our approach limitations, and future work directions in Section 7.

2 Related Work

Interactive inhabited virtual worlds received much attention these last ten years. Their creation requires several issues addressed: rendering the environment and virtual population, controlling and designing content, simulating virtual humans' behaviors and navigation, and animating them. Solutions to these issues are limited by the need for interactivity: online real-time techniques are required.

Rendering large VEs is possible by organizing scene graphs for efficient culling, or by applying level-of-detail strategies [Hamill and O'Sullivan 2003]. Rendering crowds is also feasible by exploiting a level-of-detail approach: according to its distance to the camera, a human appearance can range from a detailed 3D model to 2D pre-computed images [Tecchia et al. 2002; Dobbyn et al. 2005]. Rendering humans with pre-computed 2D images is highly restrictive as the range of possible motions is limited to the image content; generally, behaviors are limited to navigation and humans perform locomotion only. This partly explains why interactive environments made of humans exhibiting rich and varied behaviors [Shao and Terzopoulos 2005] have a limited population size.

At the global level, autonomous navigation can be seen as a path planning problem, which can be solved with different techniques. Cell-automaton methods model environments as 2D-grids [Loscos et al. 2003]. A pedestrian's global motion results from successive local motions from cell to cell, guided by probabilistic rules. These techniques are frequently used for evacuation simulations: indeed, from the statistical point-of-view, realistic results such as evacuation time, are obtained. However, such approaches fail in synthesizing high-quality smooth locomotion trajectories, due to discretization. It is also possible to decompose navigable parts of environments in other ways, *e.g.*, by using circles with varying radius [Pettré et al. 2006] or with triangulation [Lamarche and Donikian 2004]. In the case of large and complex environments, a hierarchical representation preserves fast path search times [Shao and Terzopoulos 2005; Paris et al. 2006]. In [Sud et al. 2007], authors prefer using Voronoï diagrams to solve path planning queries and agent-to-agent neighboring queries. Both cell-decomposition and Voronoï diagram-based techniques have demonstrated their ability to interactively handle large crowds. Path planning enables the user to have control over the population by distributing destination points to individuals composing the crowd.

At the local level, locomotion is controlled by steering methods. Solving interactions between pedestrians and avoiding any collision have a great impact on the motion realism. Such interactions can be solved using a set of rules [Reynolds 1987]. Helbing and colleagues make an analogy to Physics in their social forces model [2005] to represent relations between interacting pedestrians: people's accelerations result from a set of attractive (goal, friend, family) and repulsive (obstacles, danger, other people) forces. Pelechano later improved the model to avoid artifacts such as oscillations in trajectories [2005]. Treuille and colleagues [2006] made an analogy with potential fields: people move against the gradient resulting from a static field (goal) and a dynamic field (other people). In [van den Berg et al. 2008], the velocity obstacle concept from robotics is adapted to solve human interactions. Steering with level-of-detail is possible to allow large-scale populations [Yersin et al. 2008]. Recently, solving interactions from examples was achieved [Lerner et al.], [Lee et al. 2007]. However, the obtained performances are too low for applications such as interactive virtual worlds.

In conclusion, different techniques - that are efficient enough to fit interactivity requirements - are available to provide motion and action autonomy to a virtual population: action decision results from behavioral simulation, or is defined directly at design. Global navigation is possible from path planning; individuals are then steered locally in order to avoid each-other. However, several limitations remain. First, it is difficult to mix various behaviors with real-time rendering techniques. Second, online generated worlds cannot be handled, because path planning is based on specific pre-computed data structures: environment size and complexity are limited, and a fixed, known in advance geometry is required. Finally, steering human locomotion is increasingly time-consuming with the number of interactions to solve. With our approach, we first want to generate and animate virtual populations of yet unreached sizes for very large environments, out of the reach of classical crowd simulation techniques. Secondly, we want to address online generated worlds, with no a priori knowledge on their geometry. Finally, we want to help designers control the composition and behavior of virtual populations. Our solution is inspired from [Lee et al. 2006]: *motion patches* are building blocks annotated with motions. A virtual hu-

man can traverse or act in a block by using (replaying) one of the available motions. Motions are connected between different blocks by sharing common limit conditions. Blocks can be assembled to create new environments. One limitation of this approach is that no interaction between several characters evolving in a same patch is possible. We extend the concept of motion patches by creating blocks where several humans can move and interact simultaneously. The issue then grows from a geometrical problem to a geometrical *and temporal* one. A virtual population is created by assembling such blocks. As interactions are solved once for each block, and further reused during simulation, we save precious computation time. Assembling and disassembling blocks can be achieved at run-time, breaking the limitations on environment size: the virtual population is only generated in front of the spectator's point of view.

3 Principles and definitions

Our objective is to populate virtual environments by composing small pieces of precomputed animations. Such pieces, called *crowd patches*, can contain moving pedestrians, animals, or objects. Inside a patch, animations are computed so that they are cyclic over a constant period, and can be seamlessly repeated. This allows animated content to appear in endless motion. Animated objects may cross the limits of a patch and move to a neighbor patch. The trajectories of such objects must then meet common limit conditions to allow going from one patch to another adjacent one. These limit conditions are defined using the concept of *patterns*. In the following Sections, we define more precisely *patches* and *patterns*.

3.1 Patches

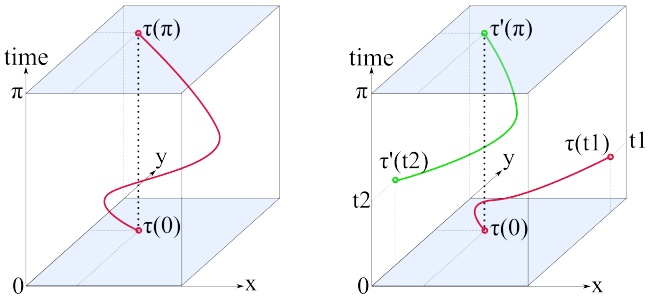

Figure 2: *Two square patches. (Left) An endogenous point performs a periodic animation in the patch: its start and end positions $\tau(0)$ and $\tau(\pi)$ are the same. (Right) An exogenous point performs a periodic motion, even if its trajectory leaves the patch at time t_1 to reappear at t_2 on the other side of the patch.*

Patches are geometrical areas with convex polygonal shapes. They may contain static and dynamic objects. Static objects are simple obstacles which geometry is fully contained inside the patch. Larger obstacles, such as buildings, are handled differently (see Section 5.3). Dynamic objects are animated: they are moving in time according to a trajectory $\tau(t)$. As previously introduced, we want all dynamic objects to have a periodic motion in order to be seamlessly repeated in time. We note π this period and we define the patch periodicity condition for each dynamic object:

$$\tau(0) = \tau(\pi) \tag{1}$$

Two categories of dynamic objects may be distinguished: *endogenous* and *exogenous* objects. The trajectory of *endogenous* objects remains inside the geometrical limits of the patch for the whole period π. An example of endogenous object is displayed in Figure 2 (left). The point's trajectory is fully contained in the patch and respects the periodicity condition (1). If the animation is looped with a period π, the point appears to be moving endlessly inside the patch. Note that static objects can be considered as endogenous objects, with no animation.

Exogenous objects have a trajectory $\tau(t)$ that goes out of the patch borders at some time, and thus, does not meet the periodicity condition (1). In order to enforce this condition, we impose the presence of another instance of the same exogenous object whose trajectory is $\tau'(t)$ (Figure 2 (right)). As the two objects are of the same type, *i.e.*, they have an identical kinematics model, their trajectories can be directly compared. Different cases are then to be distinguished:

- trajectory $\tau(t)$ is defined for $t : 0 \rightarrow T$ with $0 < T < \pi$: we impose the patch to contain another instance of the exogenous object with a trajectory $\tau'(t)$ defined for $t : T' \rightarrow \pi$ with $0 < T' < \pi$, and with condition $\tau(0) = \tau'(\pi)$.
- trajectory $\tau(t)$ is defined for $t : T \rightarrow \pi$ with $0 < T < \pi$: we impose the patch to contain another instance of the exogenous object with a trajectory $\tau'(t)$ defined for $t : 0 \rightarrow T'$ with $0 < T' < \pi$, and with condition $\tau'(0) = \tau(\pi)$
- trajectory $\tau(t)$ is defined for $t : T_1 \rightarrow T_2$ with $0 < T_1 < T_2 < \pi$. Then, condition (1) is implicitly respected, and no other instance of the exogenous object is required.

In Figure 2 (right), the point trajectory $\tau(t)$ is defined inside the patch for $t : 0 \rightarrow t_1 < \pi$. This trajectory is associated to another instance of a moving point $\tau'(t)$ so that $\tau'(\pi) = \tau(0)$. Note that $t_1 \neq t_2$ and no order relationship is required: the two instances of the moving point can be simultaneously outside or inside the patch.

3.2 Patterns

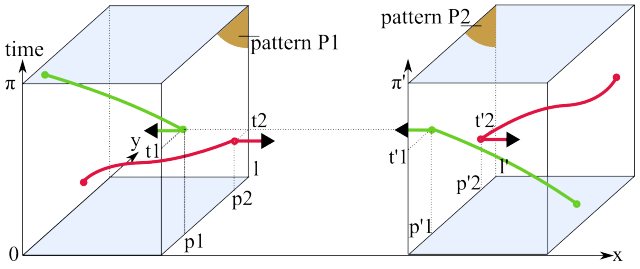

Figure 3: *Two square patches whose patterns have mirrored conditions and thus, can be connected: the point following the green trajectory can pass from the right patch to the left one seamlessly, while the point with the red trajectory moves from left to right.*

In Section 3.1, we defined exogenous objects: their trajectory exits the geometrical limits of a patch. The role of patterns is to register the limit conditions of exogenous object trajectories to allow the connection of patches. A pattern is defined for each face of a polygonal patch. As a result, patterns fully delimit patches. They are two-dimensional: a space dimension with length l, and a time dimension with duration (period) π. Patterns identify limit conditions for exogenous object trajectories. These conditions are either an input point I, or an output point O, at a specific position on the patch $p \in [0, l]$, and at given time $t \in [0, \pi]$. Thus, a pattern is fully defined from its dimension l, its duration π and a set of inputs and outputs: $P = \{l, \pi, I_i[p_i, t_i], 0_j[p_j, t_j]\}$.

We build environments and their population by assembling patches. Thus, two adjacent patches have at least one common face. They also share identical limit conditions for exogenous objects' trajectories. Indeed, when an exogenous object goes from one patch

to an adjacent one, it first follows the trajectory contained by the first patch, and then switches to the one described by the second patch. These two trajectories have to be at least continuous C^0 to ensure a seamless transition from the first patch to the second one. The patterns between the two adjacent patches allow to share these limit conditions. Let us consider the case of two adjacent patches with one common face as illustrated in Figure 3: the two patches each contain two exogenous objects going through their common face. The common face is delimited by pattern P_1 for the first patch on the left, and by pattern P_2 for the second patch on the right. we have: $P_1 = \{\pi, l, I[p_1, t_1], 0[p_2, t_2]\}$ and $P_2 = \{\pi', l', I[p'_2, t'_2], 0[p'_1, t'_1]\}$. In order to satisfy C^0 continuity for endogenous objects trajectories we must ensure:

$$\pi = \pi',\ l = l',\ p_1 = p'_1,\ t_1 = t'_1,\ p_2 = p'_2,\ t_2 = t'_2 \qquad (2)$$

We then say that P_1 is the mirror pattern of P_2. As a summary, mirrored patterns have the same duration, length, and number of mirrored inputs and outputs.

4 Creating Patches

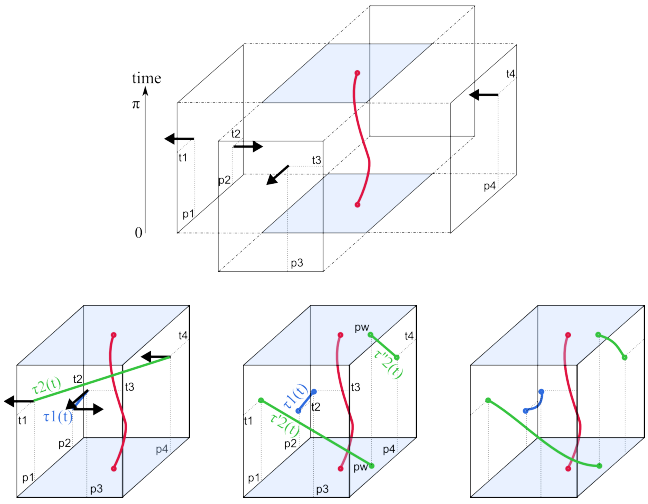

Figure 4: *Computation of exogenous trajectories in a square patch. (Top) the patterns composing the patch have 2 input and 2 output points. The patch also contains 1 endogenous pre-defined trajectory. (Bottom-left) input and output points from different patterns are randomly connected. The green trajectory is infeasible; it goes back in time. (Bottom-center) The green trajectory is split into 2 trajectories at position p_w: 2 pedestrians will use it over one period. (Bottom-right) The exogenous trajectories are updated with a particle-based method to avoid collisions between objects.*

In this section, we describe the process for creating one patch. The following section is then dedicated to the construction of environments by assembling patches. The proposed method has 3 successive steps: first, create the patch geometry and define exogenous trajectories' limit conditions by assembling patterns, second, insert static and endogenous objects, and finally, automatically compute exogenous trajectories. For an easy understanding, we provide the example of a simple square patch, illustrated in Figure 4.

4.1 Patterns Assembly

Our method for creating patches first starts by defining their shape. From the definitions of Section 3, one understands that patches are delimited by patterns: each face of the convex polygonal shape of patches is defined by one pattern. As a result, patches are created by assembling patterns. Two conditions have to be respected when assembling patterns to create a patch. First, all patterns must have a common period π which will be the patch's period too. Second, the sum of inputs defined by the set of patterns composing the patch must match the number of outputs. Indeed, each exogenous object entering the patch must, sooner or later, leave it. Obviously, if this condition is not ensured, it is impossible to respect condition (1).

An example of patch assembly from 4 patterns is displayed in Figure 4: all patterns have the same duration π and length l, and are set to delimit a square patch. Note that patterns do not require to have the same length; patches may be composed by any number of patterns, 3 at least, with no theoretical maximum. These 4 patterns also define 4 limit conditions: 2 inputs of exogenous objects $I_1[t_2, p_2]$ and $I_2[t_4, p_4]$, as well as 2 outputs $O_1[t_1, p_1]$ and $0_2[t_3, p_3]$.

4.2 Static Objects and Endogenous trajectories

The second step is either to define ourselves, or get the online information on all the static and endogenous objects contained in the patch. Static objects have their geometry fully contained in the patch. Endogenous objects are animated and have a cyclic motion with period π. In the example of Figure 4 (top), we see that one endogenous object is defined for the patch, with a red trajectory.

In the case of a patch built in a virtual shopping street, static obstacles are trash cans, trees, public benches, streetlights, signboards, etc. Endogenous objects can be humans talking, sitting, watching shop windows, etc. They can also represent animated objects or animals, such as a merry-go-round, or dogs. Note that once defined, static and endogenous objects are no longer modified. We respectively consider them as static and moving obstacles in the next step, which consists in automatically computing exogenous trajectories.

4.3 Exogenous trajectories: case of walking humans

Computing trajectories of exogenous objects is more complex than previous steps: the limit conditions of patterns must be ensured, and collision avoidance with static objects, endogenous objects (whose animations are now fixed), and other exogenous objects, must be achieved. We propose a method to automatically compute exogenous trajectories of walking humans. We consider the evolution of their global position only; a trajectory is thus modeled as a moving 2D point. Computing exogenous trajectories for walking humans is done in 3 steps. Each step is illustrated in the bottom of Figure 4.

Initialization of Trajectories. We initialize exogenous trajectories by connecting each input to an output, as defined by patterns. We thus obtain initial limit conditions for the trajectories, and count as many trajectories as the total number of inputs (or outputs) defined for the patch. Inputs and outputs are connected at random, with the sole constraint that we avoid connecting inputs and outputs of the same pattern. This way, walking humans pass through the pattern rather than head back. At first, linear trajectories are computed between a connected input $I[p_i, t_i]$ and output $O[p_o, t_o]$. We obtain for each connected pair:

$$\tau(t) = p_i + p_i\vec{p}_o.\frac{t}{t_o - t_i} \qquad (3)$$

In the example of Figure 4 (left), 2 exogenous linear trajectories are defined: $\tau_1 : I[p_2, t_2] \rightarrow O[p_3, t_3]$ in blue, and $\tau_2 : I[p_4, t_4] \rightarrow O[p_1, t_1]$ in green.

Velocity Adaptation. Inputs and outputs being arbitrarily connected, the resulting trajectories may be infeasible for walking hu-

mans: the average speed $\tilde{v} = \| p_i\vec{p}_o \| .(t_o - t_i)^{-1}$ along the trajectory may be too high, or even negative, if $t_i > t_o$, which would be non-sense. The key-idea to address this issue is to split the trajectory τ into two trajectories τ' and τ'' passing by a new way-point p_w as follows:

$$p_w = p_i + (p_i\vec{p}_o.\frac{\pi - t_i}{t_o + \pi - t_i}) \quad (4)$$

$$\tau \begin{cases} \tau' : [p_i, t_i] \rightarrow [p_w, \pi] \\ \tau'' : [p_w, 0] \rightarrow [p_o, t_o] \end{cases} \quad (5)$$

In other words, instead of having one pedestrian joining the considered input and output points, we obtain two humans: one going from the input point to p_w, while the second one heads from p_w towards the output point. These two humans have an identical position p_w at different times: at $t = \pi$ for the first human, and at $t = 0$ for the second one, thus ensuring condition (1). The new average speeds of pedestrians are respectively $\tilde{v'} = \| p_i\vec{p}_w \| .(\pi - t_i)^{-1}$ and $\tilde{v''} = \| p_w\vec{p}_o \| .(t_o)^{-1}$. If these speeds are still too high for humans, the process can be reiterated until they fall into an acceptable range (typically $[0, 2]\ m.s^{-1}$). In the example of Figure 4 (center), τ_2 is defined with $t_2 > t_3$ and is split into two trajectories.

Collision Avoidance. Trajectories may result in collisions between objects, static or dynamic. We consider static and endogenous objects unmodifiable, and refine exogenous object motions to avoid collisions. To reach this goal, we use a particle-based method that, at each time step, steers exogenous objects from a set of forces:

- objects track an attraction point moving along their initial linear trajectory $\tau(t)$
- objects are repulsed by all other objects in their vicinity.
- in order to avoid dead-lock situations, attractive and repulsive forces are balanced: the farther an object is from its attraction point and the closer t is to t_o, the more paramount attraction forces are, as compared to repulsive ones.

Particle-based simulations may result in jerky motions. We thus smooth the resulting trajectories before storing them. In Figure 4 (right), we show the trajectories obtained after the collision avoidance and smoothing steps. Note that any method could be used to update exogenous trajectories. Especially if patches are pre-computed, it is possible to use more fine-grained approaches to solve collisions. We here employ a particle-based approach for its simplicity and rapidity.

In conclusion, we described a technique for computing patches. The patch is first geometrically defined by assembling a set of patterns, which also provide a set of input and output points for exogenous objects. Then, static and endogenous objects are added to the patch. Finally, exogenous object trajectories are computed for walking humans, so that they respect limit conditions imposed by patterns, while avoiding object collisions. We describe in the following section how to assemble patches to create virtual worlds.

5 Creating Worlds

5.1 Assembly of Patches

We distinguish two major ways of using crowd patches for creating and/or populating virtual environments: a bottom-up, or a top-down technique.

The *bottom-up* approach starts from an empty scene. A first patch is created, and then, iteratively, new adjacent patches are progressively added. This process is illustrated in Figure 5 (top). Patterns

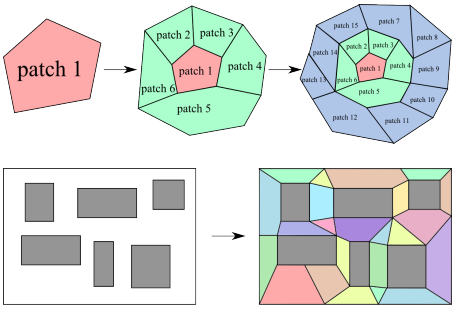

Figure 5: *Creating Worlds using (top) a bottom-up technique, e.g., procedural generation, or (bottom) a top-down approach, starting from a geometrical model of the environment.*

of a newly created patch that are adjacent to the previously inserted patches are constrained: they are directly defined by mirroring patterns of the already existing patches. Then, the patch polygon is closed by other patterns, so that they satisfy the patch assembly conditions stated in Section 4. Note how patch assembly progressively grows in an eccentric manner.

A *top-down* approach starts from a geometrical model of the environment. The obstacle-free parts of the environment are decomposed into polygonal cells that are used as a blueprint for creating patches. This technique perfectly fits the case of virtual cities where large buildings are already present, and streets have to be populated by patches. Computational Geometry provides many techniques to decompose a surface into elementary cells: some provide exact decomposition, *e.g.*, triangulation, whereas others are approximated, *e.g.*, using 2D grids. It is possible to use any of them as long as convex cells are obtained. We provide an example of a hand-made decomposition in Figure 5 (bottom).

Both techniques can be used to define the geometrical shapes of patterns. It is possible for both of them to pre-compute all patches in order to create or populate a given environment. It is also possible to generate patches on-the-fly, from the spectator's point of view: only visible parts are patched. The next step is to define the patches' content, which is dependent on the desired appearance of a given area, and is thus a design problem. Sections 5.2 and 5.3 introduce patch templates and pattern types, allowing designers to have control over the environment content.

5.2 Patch Templates

The content of a patch is dependent on its precise location in the environment, and on the considered environment itself. Let us take the example of a shopping pedestrian zone. The corresponding patches should then contain static obstacles such as benches, trees or flowerpots. Endogenous objects can be people standing in front of shop windows, talking together, sitting on benches, while exogenous objects are simply walking humans.

Designers want to have a certain control over the environment content, but accurately defining the objects in each patch is too time consuming. A solution is then to automatically generate patches from given templates. A *patch template* groups patches meeting a set of constraints on their objects and patterns. In order to associate geographic zones with desired templates, designers provide a *template map*. The map defines which template to use at any point of the environment. We can also address the specific case of environments that are modeled on-line in real-time, using procedural methods. However, they need adaptation to generate the template map in parallel with the geometry.

When a patch of a given template is created, some static and endogenous objects are randomly selected among the allowed set to compose its content. Designers also need to control the flow of walking humans going through patches, which implicitly defines the overall distribution of people in an environment. This is possible by constraining the pattern types to use.

5.3 Pattern Types

Controlling walking humans in our method is achieved by defining specific pattern types. These types allow to choose the specific distribution of input and output points in space, or time. We give here some examples of usage for specific distributions.

Empty Patterns. Environments are likely to contain large obstacles, such as buildings (as seen in Figure 5 (bottom)). In order to avoid collisions between exogenous walking humans and these obstacles, patterns delimiting them are empty of inputs or outputs. Indeed, a patch with an output at an obstacle border would result in an irremediable collision, whilst an input would result in a sudden apparition of a human through the wall.

One-way Patterns. One-way patterns are exclusively composed of inputs or outputs. When correctly combined in a patch, they allow to simulate, for example, one-way flows of walking humans.

Specific I/O Space Distribution. It is possible to limit the range of input and output positions in space on a given pattern. This allows to simulate the presence of a narrow passage, such as a door, between two patches for instance, or to simulate crowded streets of walking humans forming lanes.

Specific I/O Time Distribution. Users may want pedestrians to enter or exit patches irregularly in time. For instance, at zebra crossings, pedestrians leave sidewalks when the walk sign is green, and have to reach the opposite sidewalk before the light switches to red. Such temporal conditions can be respected by constraining the patterns' inputs and outputs in time instead of randomly selecting them within the period π.

In conclusion, we have introduced the concepts of patch templates and pattern types. A template provides control over the environment content by inserting various objects, static or moving, according to the designers' will. Pattern types gives control on the flows of walking humans. Templates and types are user-defined. In Section 6, we put these concepts into practice and present our results.

6 Applications and Results

In this Section, we illustrate the concept of crowd patches with two applications. First, we procedurally generate a potentially infinite pedestrian street, populated with a bottom-up approach. Second, we use a pre-computed city environment complemented with a template map to populate it. Finally, we analyze the performances obtained for the patch generation and the online simulation.

6.1 Applications

Bottom-Up Approach. The main advantage of a bottom-up approach is that environments are procedurally generated at run-time and can grow in unexpected ways. We illustrate this approach here with a simple example: a potentially infinite pedestrian street, procedurally generated at run-time with a rule-based algorithm. The infinite deployment of this street is demonstrated in the companion video and in Figure 1 (left).

To use crowd patches in a bottom-up approach, several steps have to be followed in a pre-process. First of all, we design a library of static obstacles: streetlights, trash cans, trees, city maps, benches, etc. These static objects are illustrated in Figure 6 (left). Secondly, endogenous objects, like humans talking together, playing children, or seated people, are created (Figure 6 (center)). In a third step, the required pattern types are identified: an endless street requires among others, empty patterns for building borders, and specific I/O space distributions to simulate lane formations in highly crowded portions of the street. For each identified pattern type, we then generate various patterns. Note that the set of patterns does not need to be exhaustive; if a pattern is missing, it can be generated at run-time. However, the smaller the number of patches/patterns to generate online, the faster the simulation. Various pattern examples are illustrated on the right of Figure 6. Similarly to pattern types, patch templates are identified according to the sort of environment to generate online. A first non-exhaustive set of patches for each template is created too, using the pattern library. Some of these patches are shown in the bottom of Figure 6. Finally, to further vary the patch content, we also define an additional set of grounds, or textures that can be applied on a patch: cobblestones, grass, asphalt, etc. The resulting variety is visible in the video and Figure 1.

At run-time, patches are introduced in the scene wherever the camera is looking. Specific patches are first looked for in the existing library. If no patch matches the requirements, it is generated on-the-fly. If the patch generation requires a specific pattern that has not yet been created, it is also created online. A second important step at run-time is to update each pedestrian on its trajectory. When a human reaches a patch border, we seamlessly make it move to the neighbor patch. To efficiently achieve this, each trajectory possesses a parameter pointing to the neighbor trajectory that should be followed next.

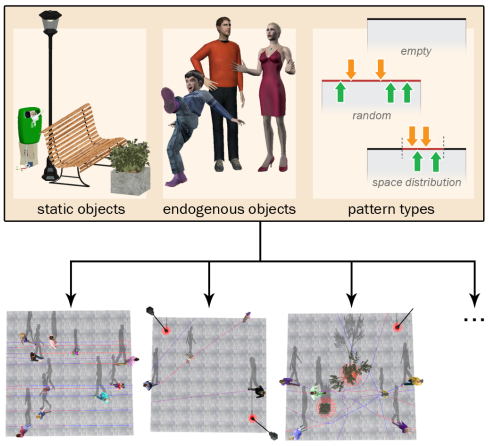

Figure 6: *To build a lively pedestrian street, we use (left) static obst., (center) endogenous obst., and (right) specific patterns. (Bottom) From these sets, we first define several patch templates. Finally, each template is instantiated to result in a variety of patches.*

Top-down approach. A major advantage of pre-defined environments is the possibility to pre-compute many elements and dedicate the resources spared in this way for run-time rendering and animation of pedestrians. Our second example, illustrated in Figure 1 (right) and in the companion video, is a large city whose geometry has been previously designed, along with a template map to assist the populating process.

In such a case, the whole set of patches can be computed offline. Based on the template map, each patch is consecutively generated, along with the required patterns. Note that in the case of a city, we typically require patterns of all types to simulate streets, building entries, public places, etc. Once the full map has been computed, the run-time execution can start. Since everything is pre-computed, the user can move in the whole city and observe the pedestrians evolving everywhere. At each time step, the remaining work is reduced to pedestrians' animation, navigation, and rendering. We further detail the achieved performances in the next Section.

6.2 Results

The performance tests and the video have all been performed on a desktop PC with a dual core 2.4GHz, 2GB of RAM, and an Nvidia Geforce 8800 GTX. We have instantiated 6 different human templates, rendered with a level-of-detail approach, ranging from about 6,000 to 1,000 triangles. They are animated with a motion capture database. Shadows are computed on the environment and the humans. In our implementation, the computation of a patch takes approximately 60ms. This number fits perfectly to real-time procedurally generated environments, given that a first non-exhaustive library of patches has been pre-computed.

The infinite street environment has been built with one template patch, and a total amount of 10 patterns only, of empty, space-constrained, and random types. The patterns all have a period of $10s$ and an $8m$ length . The average number of visible patches in front of the camera is 500, 75% of which are buildings with 2 entrances where people come in and out. The remaining visible patches represent the street and its crossings. There are approximately 750 visible walking humans at all times (exogenous objects), 30 idle humans representing endogenous objects, and 80 static objects, such as benches, or streetlights. On average, we obtain 30 frames per second, including the rendering of the humans and the environment. The frame rate is relatively constant over the whole progression on the street, since the number of patches and humans which are computed and displayed remains the same. As illustrated in the video, the result is impressively varied, given that we have used 10 patterns only. In Table 1, we show the variation of the frame rate for the simulation only (rendering is deactivated), when the number of patches to update is modified.

# of displayed patches	# of displayed humans	average frame rate
1	1-5	275-285
45	67-93	140-155
103	144-206	135-140
206	372-420	130-135
411	586-634	110-120
810	1185-1240	95-105
1845	2596-2693	40-45
3772	3572-3725	30-35

Table 1: *Frame rate evolution (without rendering) for a varying number of patches to simulate.*

The city environment was pre-computed with 20 different patterns of a $10s$ period and an $8m$ length. All the types introduced in Section 5.3 were represented. The template map that was used has a size of 32x32 pixels, each representing one patch. We used 4 patch templates (city place, street, building entrance, public park). Open spaces represent 25% of this map. We have simulated approximately 2,500 exogenous humans, 180 endogenous humans, and 300 static objects at 20fps (including rendering). In this case, the trajectories have all been pre-computed. The obtained frame rate is thus limited by the rendering of the whole city and its citizens, rather than the simulation itself, which only takes 5-10% of the resources. With this second example, we confirm that a small number of patterns can already bring a great variety of trajectories to the environment.

7 Conclusion, Discussion and Future Works

We presented a method for populating large-scale interactive virtual environments with walking and idle humans, as well as animated and static objects. The key-idea of our solution is to build environments from a set of blocks, the crowd patches, that describe periodic motion for a small local population, as well as other environment details. Periodicity in time allows endless replay. Our solution provides an advantageous trade-off between memory usage, computation needs, and motion quality. Crowd patches allow to handle large-scale environments and to densely populate them with believable motions. Our method also provides a solution for designing environments from sets of templates. Our approach has some limitations. However, there are solutions to alleviate them, as discussed in the following paragraphs.

Time Period. Patches and patterns have a user-defined period. Contiguous patches need to have an identical period to be connected. As a result, a limitation of our solution is the need for an identical period for all interconnected patches. Changing the period is easy when dealing with exogenous objects, as we proposed an automatic technique for computing their periodic animation. However, endogenous objects may have hand-designed animation, making the change of period duration more difficult.

Static Patches. Patches are created once at a given place: the overall distribution of the population in the environment is thus static in time. As a result, it is not possible to synthesize dynamic events, such as a small demonstration of people going through the environment. Also, due to the finality of the patches' layout, no goal-directed navigation can be achieved for a human instance: by following a specific pedestrian, a spectator may observe some incoherent navigation (roaming or going nowhere). To solve these limitations, an interesting direction for future work would be to allow dynamic changes of patterns and patches.

Variety in Space and Time. If users' point of view remains static for a long period of time, the animation periodicity may be detected. It is possible to make the detection more difficult by generating a variety of patches from identical patterns, instead of a unique instance. This variety can be obtained by randomizing initial connections between inputs and outputs at the patch creation stage, whilst endogenous and static objects remain. It is also possible to imagine further variations when computing trajectories by modifying the steering method parameters when solving interactions between people. However, memory consumption is proportionally increased. Generating a variety of patches for identical patterns appears to be the most interesting direction for improving our approach.

Interactivity. As emphasized in the two previous paragraphs, character motions are precomputed and fixed using crowd patches. This prevents rich interactivity between the virtual population and the user. We add subtle details, such as humans looking towards the camera, that simulate interactivity. However, the pre-computed locomotion trajectories do not account for the presence of spectators. We propose two future directions to allow interaction between the users and the virtual population. The first one would be to locally

edit locomotion trajectories in order to account for the presence of spectators. However, patterns define strict limit conditions for these trajectories. Managing editing and limit conditions would then need special care. A second solution would be to consider that patches are only used to simulate secondary characters, whereas interactive digital actors would be added to this background population. Interactions between digital actors and secondary characters would also need careful management.

Improving Motion Quality. Exogenous trajectories for walking humans are computed using a particle-based simulation. Such a technique is advantageous, because it very efficiently computes trajectories that remain reasonably believable. Recently, Kwon and colleagues [2008] proposed a solution to edit the motion of groups. To further improve our results quality, we could automate and use such editing techniques to compute trajectories inside patches and combine them with small group motions. As space and time editing is achievable with Kwon's approach, group trajectories can be adapted in order to meet limit conditions imposed by patterns.

In conclusion, crowd patches have several paramount advantages. We ease the design process for virtual populations using a library of patch templates. Secondly, we make it possible to use time consuming animation techniques, as motions are pre-computed and stored. In the case of environments and patches generated on-line in real-time, our solution is more efficient than a complete crowd simulation. Finally, we allow the handling of large-scale environments and populations by drastically lowering the need for computation resources dedicated to simulation. Another major advantage of our approach is the ability to guarantee simulation content: trajectories are fully solved once patches are computed, and the resulting animation is reproducible: risks of deadlock situations or other artifacts are eradicated, and the simulation can be subtly locally edited (*e.g.*, by replacing some patches) at some given places without any impact on the remaining parts of the environment. Some drawbacks relative to the repetition of precomputed motions have been pointed out in the previous paragraphs. However, several tracks have been proposed to alleviate these limitations.

Acknowledgements

The authors would like to thank Mireille Clavien for her exceptional designing work. Special thanks to Marc Christie for fruitful discussions and valuable comments. Thanks to Helena Grillon and Benoît Le Callennec for proof-reading this paper. This research is sponsored by the Swiss National Research Foundation.

References

DOBBYN, S., HAMILL, J., O'CONOR, K., AND O'SULLIVAN, C. 2005. Geopostors: A real-time geometry/impostor crowd rendering system. In *I3D'05: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 95–102.

HAMILL, J., AND O'SULLIVAN, C. 2003. Virtual Dublin a framework for real-time urban simulation. *Journal of WSCG 11*, 1.

HELBING, D., BUZNA, L., JOHANSSON, A., AND WERNER, T. 2005. Self-organized pedestrian crowd dynamics: experiments, simulations and design solutions. *Transportation science*, 1–24.

KWON, T., LEE, K. H., LEE, J., AND TAKAHASHI, S. 2008. Group motion editing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, 1–8.

LAMARCHE, F., AND DONIKIAN, S. 2004. Crowds of virtual humans : a new approach for real time navigation in complex and structured environments. *Eurographics'04: Computer Graphics Forum 23*, 3 (September), 509–518.

LEE, K. H., CHOI, M. G., AND LEE, J. 2006. Motion patches: building blocks for virtual environments annotated with motion data. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, 898–906.

LEE, K. H., CHOI, M. G., HONG, Q., AND LEE, J. 2007. Group behavior from video: a data-driven approach to crowd simulation. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 109–118.

LERNER, A., CHRYSANTHOU, Y., AND LISCHINSKI, D. Crowds by example. *Eurographics'07: Computer Graphics Forum 26*, 3.

LOSCOS, C., MARCHAL, D., AND MEYER, A. 2003. Intuitive crowd behaviour in dense urban environments using local laws. In *TPCG '03: Proceedings of the Theory and Practice of Computer Graphics*, 122.

PARIS, S., DONIKIAN, S., AND BONVALET, N. 2006. Environmental abstraction and path planning techniques for realistic crowd simulation. *Computer Animation and Virtual Worlds 17*, 325–335.

PELECHANO, N., O'BRIEN, K., SILVERMAN, B., AND BADLER, N. 2005. Crowd simulation incorporating agent psychological models, roles and communication. *First International Workshop on Crowd Simulation*.

PETTRÉ, J., DE HERAS CIECHOMSKI, P., MAÏM, J., YERSIN, B., LAUMOND, J.-P., AND THALMANN, D. 2006. Real-time navigating crowds: scalable simulation and rendering: Research articles. *Computer Animation and Virtual Worlds 17*, 34, 445–455.

REYNOLDS, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, M. C. Stone, Ed., vol. 21, 25–34.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 19–28.

SUD, A., ANDERSEN, E., CURTIS, S., LIN, M., AND MANOCHA, D. 2007. Real-time path planning for virtual agents in dynamic environments. *Proceedings of IEEE VR*, 91–98.

TECCHIA, F., LOSCOS, C., AND CHRYSANTHOU, Y. 2002. Image-based crowd rendering. *IEEE Computer Graphics and Applications 22*, 2, 36–43.

TREUILLE, A., COOPER, S., AND POPOVIC, Z. 2006. Continuum crowds.

ULICNY, B., DE HERAS CIECHOMSKI, P., AND THALMANN, D. 2005. Crowdbrush: interactive authoring of real-time crowd scenes. In *ACM SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of individual agents in crowded environments. *Symposium on Interactive 3D Graphics and Games (I3D)*.

YERSIN, B., MAÏM, J., AND THALMANN, D. 2008. Real-time crowd motion planning: Scalable avoidance and group behavior. *The Visual Computer 24*, 10, 859–870.