EUROGRAPHICS 2004 / M.-P. Cani and M. Slater
(Guest Editors)

Volume 23 (2004), Number 3

Scalable behaviors for crowd simulation

Mankyu Sung, Michael Gleicher and Stephen Chenney

Department of Computer Sciences
University of Wisconsin – Madison

Abstract

Crowd simulation for virtual environments offers many challenges centered on the trade-offs between rich behavior, control and computational cost. In this paper we present a new approach to controlling the behavior of agents in a crowd. Our method is scalable in the sense that increasingly complex crowd behaviors can be created without a corresponding increase in the complexity of the agents. Our approach is also more authorable; users can dynamically specify which crowd behaviors happen in various parts of an environment. Finally, the character motion produced by our system is visually convincing. We achieve our aims with a situation-based *control structure. Basic agents have very limited behaviors. As they enter new situations, additional, situation-specific behaviors are* composed *on the fly to enable agents to respond appropriately. The composition is done using a probabilistic mechanism. We demonstrate our system with three environments including a city street and a theater.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

1. Introduction

Crowds are an important feature of the real world, and hence high quality crowd simulation is vital for many virtual environment applications in education, training, and entertainment. There are several conflicting goals in crowd animation. Simple characters are more efficient to evaluate, but complex characters can capture more realistic crowd behaviors. Authors need a control over the actions of the crowd, but we cannot control every agent individually. Finally, crowds for virtual environments must be visually plausible. *Scalable* simulation is required to address these conflicting goals, particularly with respect to the complexity of the environment and the wealth of behaviors required by the crowd. In a scalable simulation, the complexity of the characters and their control grows slowly as the environment demands more of them, and visual quality is always retained.

In this paper we present a novel approach for simulating a crowd in a complex virtual environment that is scalable, directable and visually convincing. In this approach the crowd is simulated on two levels (Figure 1). At the high level, we adopt a *situation-based* distributed control mechanism that gives each agent in a crowd specific details about how to react at any given moment based on its local environment. At the low level we use a probability scheme which computes probabilities over state transitions and then samples to move the simulation forward. In addition, we use a *Snap-Together-Motion* [GSKJ02] process to maintain visual motion quality. This ensures that there are no artifacts as motions are concatenated together over time.

The main inspiration for our approach is the anonymity offered by crowds. When we look at a crowd, we care only about *what* is happening, not *who* is doing it. This has two implications. First, the actions of the crowd should be driven by situations – what is happening – and not by individuals. This motivates our situation-based strategy. Secondly, viewers of a crowd cannot individually identify and track agents. Rather, viewers are attentive to overall statistics of the crowd: the direction it's moving, the apparent agitation of its members, the number of people waiting for the bus. Hence, the actions of an individual matter only in its short-term contribution to the crowd's behavior, and not in their long-term planning. This motivates our probabilistic approach to simulation.

A key observation is that a given character is never in more than a few situations at once. This limits the set of behaviors a character requires at any given moment. For example, while watching a movie an agent needs to know how to sit in a theater, but it doesn't need to remember how it bought

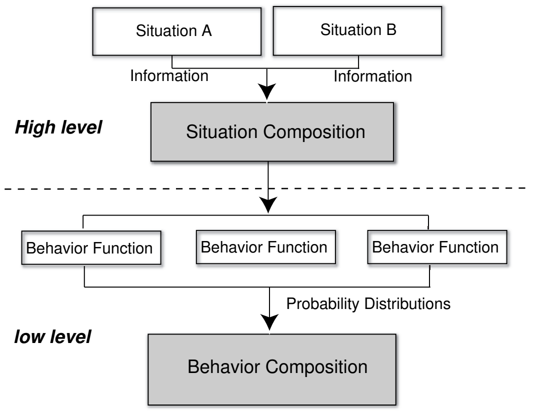

Figure 1: *An overview of our two-level agent architecture. At the high level, situations control the events that agents respond to, and provide mechanisms for handling the events. At the low level, the behaviors resulting from multiple events are composed with a probabilistic technique to choose the next action.*

the ticket, nor how it uses the bathroom. We exploit this with *composable* behaviors. An agent starts off with simple behavior models. When it enters a situation, such as approaching a ticket booth, it is augmented with the machinery necessary to purchase tickets. Once a ticket is purchased, the machinery is removed again.

The first advantage of this approach is ease of authoring; it breaks the problem of agent design into the design of local activities, rather than one monolithic system. Second, reuse is enhanced; the core behavior and actions can be used in any environment and the situation specific modules can be shared wherever situations are shared. Third, efficiency is improved; at any given moment an agent only has information for the situation that they are in – not all of the information for the entire environment. Finally, de-centralized control of agents makes overall performance scalable, especially when a large crowd is simulated; each situation takes care of only a small number of agents.

The situation-based approach puts an emphasis on environment design because overall crowd movement depends on the situations present in the space and where they are located. For specifying situations, we adopt a *painting interface*. It allows the user to specify a particular situation by drawing it on the environment directly. Situations can be easily composed by overlaying several in one area.

The next section provides an overview of our approach, followed by a review of related work. Section 4 explains the probabilistic simulation scheme. Section 5 presents the details of our situation-based approach. We close with several experiments and discussion of future work.

2. Overview

The aim of our simulation is to produce the moment to moment actions of individuals in a crowd. At a given instant in time, an individual may perform any number of actions, but in our system the set is limited to those that we are able to animate. Because we use motion synthesis based on connecting clips of existing motion (Snap Together Motion [GSKJ02]), the individuals have a finite repertoire of actions (the available clips), and therefore at any instant they will have a discrete set of choices for what action to do next. The sequence of such discrete choices represents the behavior of an agent over time.

Snap Together Motion constrains the set of transitions from one state to the next (for now, consider a state to be a position and orientation) to the small set of movements that can plausibly follow the current state. The set of actions forms a directed graph where the nodes represent states where different choices are available, and the edges represent the actions that can be performed in that state. The use of a directed graph, or finite state machine, is a common mechanism for synthesizing the movements of characters in interactive systems.

The *behavior* of an individual character is its sequence of actions. A particular sequence stems from a set of choices: in each state, the character must choose which state to move to next. An *action selection method* is an algorithm for making the choices. Changes in the method give a character distinctive behaviors. Our challenge is to create action selection methods that work for crowd simulation and meet our goals of authorable and scalable behaviors.

We exploit the observation that when individuals are anonymous their specific actions may appear somewhat random. Consider a man crossing a busy city street at a particular instant. There are many actions he may choose: he might continue walking across the street; he might turn around and walk back the way he came; he might glance to the side to check to see if a car is disobeying the traffic signals. If we knew the individual, his choice might be clear: we might know that a particular person is prone to remembering that he forgot something in his office while crossing the street. For an individual we know little about (a member of a crowd), we cannot say for certain. We thus model the action choice probabilistically; a person crossing the street is more likely to keep crossing, but there is some chance that he may turn around and walk back.

When a new state is required, our probabilistic action selection method considers the choices that the character has for its next state, creates a probability distribution as to which choice is likely to be taken, then samples from this distribution to determine the course of action. The challenge, then, becomes determining the probability distributions such that the sequence of action choices leads not just to plausible behavior of the individual, but also the desired behavior of the entire crowd.

Complex behavior (and a correspondingly complex distribution function) is often made from a number of simpler pieces. For example, a person walking in the city will be avoiding others, trying to move towards a goal, trying to obey traffic laws, and so on. We define *behavior functions* that describe the actions for each of these simple behaviors through probability distributions. We compose distributions to create the final action selection method. In this way we combine simple functions into more complex aggregate behaviors. Behavior functions are discussed in detail in Section 4. A behavior function may depend on many things. For example, it may depend on the location of the individual (the middle of the street is not a good place to stop), what the agent is able to "sense" (wait until the signal says "walk" before crossing), or even an aggregate controller that attempts to regulate the crowd (if there are too many people on one side of the street, it is more likely for an individual to cross so that things are better balanced).

Our situation-based approach determines which behavior functions are currently influencing a character, and hence determines the overall behavior of each agent. When a character enters a situation, such as crossing the street, our system extends the character to enable appropriate behavior. Primarily, it adds temporary behavior functions to the character that are composed with its existing ones, allowing it to choose its actions more wisely. Situations may also add actions to the character - an individual need not know how to look both ways unless she or he is crossing the street. In fact, even the character's ability to sense its environment can be adapted to the situation. Only when in the street crossing situation does the character need to know how to sense the status of the crossing signal. When the character leaves the situation, all of the situation-specific extensions are removed. Section 5 describes the details of extending the behaviors of agents.

3. Related Work

For the general purpose of generating human-like behaviors, many researchers have proposed cognitive agent architectures [FTT99, WCP*, AC01] that are comprised of a knowledge representation, algorithms that learn new knowledge, and modules that plan actions based on the knowledge. These systems are not scalable because their complexity grows at least as rapidly as the overall environmental complexity.

Rule-based schemes, such as Reynolds' *boids* models [Rey87] are fast enough for use with large numbers of agents. Commercial systems, such as *SoftImage/Behavior*, *AI-Implant*, *Character Studio 4.0* and *Massive* [Koe] also appear to be rule-based. These systems are not scalable from an authoring perspective. Creating crowds for complex environments is extremely time-consuming and error prone. Real-time systems of the type we are concerned with cannot touch-up poor behavior in the way that offline film production can. Hierarchical schemes have been proposed to address scalability [FMS00, MT01]. In particular, Musse et. al. endows crowds with different levels of autonomy for hierarchical crowd behaviors [MT01], but complex individual behaviors have not been demonstrated.

At the other extreme, physics inspired approaches, such as a social force models [HM95, HFV00] or particle systems [EE97, GLM99, BMdB03], can create realistic crowd flow but are only applicable to situations such as emergency evacuations, in which crowd behavior is limited and interactions with the environment are minimal.

In order to reduce the complexity of controlling crowds yet retain detailed behaviors, several systems [FBT99, TLCC01, FMS98] have attached information to the environment to guide the characters within it. Simple examples include driving and pedestrian simulators that embed lane or pathway information in the model. Our approach also embeds information, such as planned paths, into the environment, but, in addition, we include the *behaviors* to interpret that information.

The most commercially successful system of this type is the computer game, *The Sims* [FW01], in which objects advertise services, such as "satisfy hunger", and define a procedure that is run when the character responds to the service. *The Sims* highlights the authoring advantage of a rich environment: part of the game's appeal is the ability to easily add new objects and have characters respond to them. *The Sims* add behaviors by enforcing a specific, linear *plan* to the interacting agents. If the desired object-specific behavior is not amenable to a simple plan, the approach breaks down. For instance, a "mingle with a crowd" behavior could not be added. Such interactions must be part of the characters innate behavior, increasing its complexity. In contrast, our method adds "behaviors" that interact on equal terms with the existing behaviors and have all the power of rule-based state machines. This allows for much simpler underlying characters and more complex extension behaviors.

Similar to *The Sims*, Kallmann and Thalmann [KT98] describe *smart objects*: objects that provide a plan for their use. The character, upon approaching an object, is told to execute a specific sequence of steps. For instance, an elevator informs a character to push the button, wait, then enter when the door opens. This approach is similar to ours in that the characters are provided all necessary information from the environment directly. However, in our approach the situation is more general concept that includes non physical effects such as friendship among characters. Moreover, rather than indicating a specific behavior to a character at a particular time, our situations propose a composable behavior that can be combined with others. One result of this is that characters in our system do not always respond to the same object in the same way, just as real people behave.

Robotics algorithms have been applied to animate multiple characters. Bayazit et. al. [BLA02] uses a global road map to set collision-free paths for multiple characters, and similar methods have been introduced [Feu00]. Tsi-Yen Li et. al. proposed the Leader-Follower model [LJC01], the main goal of which is to generate collision-free paths for characters. Since their focus is on path planning for multiple characters, complex behavior such as "find an empty seat and sit down, then watch a movie" cannot be simulated. In our approach, collision avoidance is achieved through a behavior function that penalizes states which may cause a collision by giving low probability to them.

We need good motion data to produce visually plausible crowds. Many example-based motion synthesis techniques based on blending have been proposed [RCB98, KG03]. They typically interpolate several motions to synthesize a desired new motion. For example, to get a walking motion with a particular speed, similar walking motions are found in the motion set and interpolated to get the desired walk. In our system, on the other hand, we use appropriate motions directly without any interpolation because we assume that a behavior can be represented with a series of motions. Therefore, to simulate a particular behavior, we only need to determine which motions are appropriate and *select* one, rather than creating a new motion. Performance is improved with this approach because we don't use search and interpolation.

Tecchia et. al. [TC00, TLC02] describe a real-time, image-based alternative to motion-captured data for crowds. However, the realism of image-based approaches is limited by the amount of imagery that must be stored in order to handle arbitrary, close-up viewing conditions.

4. Probability Scheme

As described in Section 2, the behavior of an agent comes from its choices about which actions to take. At each time step of the simulation, the agent has a state $s = \{t, \mathbf{p}, \theta, a, \mathbf{s}^-\}$, where t is the time, $\mathbf{p}$ is a 2D position vector, θ is an orientation, a is an action and $\mathbf{s}^-$ is a list of previous states. The position and the orientation indicate the spatial disposition of the character. The action is directly linked to a motion clip and determines which clip should be played for the current frame. The past states are used by the behavior mechanism to give some correlation in the agent's behavior over time. Without some previous state information it is difficult to enforce behaviors like "walk in a straight line". The aim of the probabilistic state selection mechanism is to choose a next state given the current state.

The link between actions and motions means that we have a finite set of possible choices for the next action – it has to be one of the available motion clips. Each potential next state has an associated probability representing the chance of being selected. Our behaviors modify these probabilities on the fly, as well as the set of potential states. Note that this is conceptually similar to probabilistic finite state machines typically used for character AI [WP01], but in our approach both the state graph and the transition matrix are modified at run time. Our state transitions also represent lower level behavior compared with traditional finite state machines.

4.1. Behavior functions and behavior composition

When examining the movement of a crowd in real life, we easily see that there are many different factors that influence the behavior of an agent. For example, people might change their route depending on whether or not there is a person or object nearby. Or, if they have a target place that they are moving toward, they usually go as straight as possible. To simulate crowd behavior realistically, we need a way to take account of these various kinds of factors and synthesize a complex behavior that reflects them. More specifically, given the possible next states, we need a way to judge these states from the point of view of different factors and then compose transition probabilities to reflect all factors.

Each influence on the agent is encoded in a *behavior* and special functions called *behavior functions* perform the task of transforming a behavior into a set of transition probabilities on states. For example, the "overlap behavior function" takes care of avoiding other agents. The "Don't turn behavior function" checks if a state transition causes too much change in orientation. The behavior functions judge the potential state transitions with their own rules independently and return the probabilities.

Suppose the set of possible next states is $\mathbf{S} = \{s_1, s_2, ..., s_m\}$, where s_i is a particular state. Note there are m states. A behavior function, k, evaluates all states in S and calculates their probabilities. A prototype behavior function is shown below.

```
Behavior function $k(States\, S[], Prob\, P[])$
{
  For state $s$ in set $S$ do:
      x = evaluate($s$)
      $P_k(s)$ = sigmoid($x, \alpha$)
}
```

The *evaluate* routine inside a behavior function is a general function that can use any information available to it, such as the state of various features of the environment or the past state of the agent or distance from a position in the environment. The evaluation function characterizes a conceptual notion of "behavior" that the behavior function is trying to model. Considering these information, the *evaluate* function returns any real value. Then, the value is normalized to the range $[0, 1)$ using a *sigmoid* function:

$$P_k(s) = \frac{1}{1 + e^{-\alpha x}} \tag{1}$$

We do this to make composition of behaviors more stable. The constant α determines the slope of the curve and be chosen differently for each behavior function. The values we use range from 1.0 to 10.0.

The primary reason for using behavior functions is that we are able to compose several component behaviors to synthesize more complex behaviors that embody all the influences of the component behaviors. Composition of behaviors is

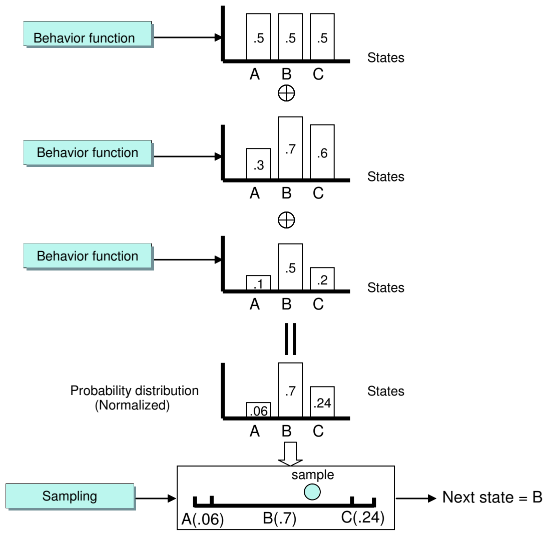

Figure 2: *The behavior functions compute the probability of input states independently. The probability distributions that are computed from behavior functions are then composed. Finally, the next state is selected through sampling on the composed probability distribution.*

simply the multiplication of the probabilities the behavior functions produce (see Figure 2). That is,

$$P'(s) = \prod_{k=1}^{n} P_k(s)$$

where n is the number of composed behavior functions, s is a possible next state, the $P_k(s)$ is a probability distribution from k^{th} behavior function and $P'(s)$ is an unnormalized probability distribution function. We use multiplication because it allows one behavior to veto a state transition (by setting its probability to 0). Finally, re-normalization should be performed on the final composed probability distribution to ease sampling. For each state s_i:

$$P(s_i) = \frac{P'(s_i)}{\sum_{j=1}^{m} P'(s_j)}$$

Once we have a final probability distribution, the simulation algorithm chooses the particular state transition to perform by sampling according to $P(s)$. The sampling allows the character to choose not only states with high probability but also states with low probability, even though it's not frequent. This gives the crowd an element of randomness.

4.2. Default states and behaviors

If an agent is not in any specific situation, we would still like it to exhibit certain default actions. This relieves the user from having to specify a situation for every point in the world. The default state transitions in our system are the identity transition (which does not change the agent's state), five walking actions with different turn angles and six turning in place actions with different angles. Each transition takes a character from one position and orientation in the environment to another, and has an associated motion clip.

Several different default behavior functions are combined to produce a sequence of states appropriate for a character wandering through an environment.

- ***ImageLookup*** **Behavior** : This behavior gets a bitmap describing the obstacles in the environment, and gives zero probability to states that cause the character to enter a place where some objects are located. Otherwise, it gives high probability.
- ***TargetFind*** **Behavior** : If the character has a goal position, this behavior gives high probability to states that make the character move toward the position. Otherwise, it gives low probability. Basically, this behavior is for moving characters from one place to another through path planning. We use the Probabilistic Road Map (PRM) method [KL94, OS94] in which many way-points are distributed randomly in the environment and linked together with Dijkstra's all-pairs shortest path algorithm including visibility. Since we compute all-pairs paths between two way points in a preprocessing step, at run-time we can find any path in real time.
- ***Overlap*** **Behavior** : This behavior gives zero probability to states that cause a collision with another character. Otherwise, it gives high probability. The collision detection algorithm used in our system is a simple adaptive spatial subdivision.

These default behaviors are always composed unless an agent is in a situation that specifically ignores them (see Section 5.2).

5. Situation and pluggable character architecture

Our basic assumption is that an environment consists of a set of different situations and only a few characters are under the same situation at the same time. A situation is distinguished from other situations by what typical behaviors the crowd can show. That is, the situation in our system controls the behaviors of a local group of agents. This provides distributed control over the crowd because we can give situation-specific behaviors to characters only when they need them. In addition, our approach provides composability among multiple situations so that agents can respond to the complex scenario when several different situations are combined.

A situation can be any circumstance that has typical local behaviors. From observations of real people, we easily see that one of the most important factors that affect agent behavior is its spatial location. For instance, a behavior for getting on or off a bus can be seen only at the bus stop. Therefore, we can set the "bus stop" situation around the bus stop. The relationship between people is another important factor in determining behavior. Friends usually walk together when they move. Therefore, "walking-together" is a typical behav-

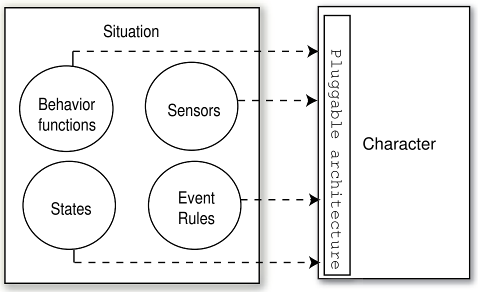

Figure 3: *When an agent is under a situation, it gets all the information from the situation directly. This includes states, behavior functions, sensors and event rules.*

ior that can be seen among people with friendship. It means that "friendship" can be another situation. We categorize all situations into two different kinds.

- **Spatial situation**: The situation has a region that it affects in the environment (e.g, bus stop, ticket booth, ATM machine). This situation cannot be moved after definition (which can be at run-time) but can be deleted at run-time.
- **Non-spatial situation**: This situation (e.g friendship, group) does not have any region in the environment because it is attached to the crowd directly, not to the environment. Therefore, this situation can only be set at run time and agents are explicitly added or removed from the situation by the user.

The structure of a situation is shown in Figure 3. The behavior functions implement behaviors specific to the situation, the sensors provide sensing capability for characters to catch events, states are state transitions (motion clips) that are used only for the local behaviors, and the event rule is a way to relate events to specific behaviors. When a character is under a situation, all these components are added to its representation at the same time. For a spatial situation, the components are added when the character first enters the situation's region of influence. For non-spatial situations, the character is affected when the user assigns the situation to it. The components are removed from the agent when it leaves the situation's region or the situation is otherwise removed. This dynamic adding and removal of behaviors enables us to achieve scalable agents.

The first thing a situation does to an agent under its influence is extend its states by adding new transitions to the agent's current state transition graph.We refer to this as *extending* the graph. At the same time, this increases the number of states available for selection by the probabilistic selection mechanism. An example of an extended graph is shown in Figure 4. In essence, each situation has its own small state transition graph that is grafted onto the agents existing graph by connecting new transition edges. Each situation knows where in the agent's default graph its own small state graph

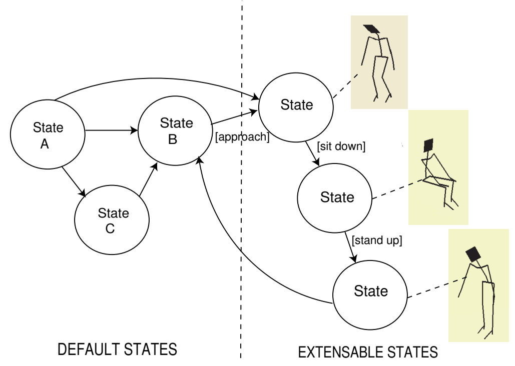

Figure 4: *The states are organized as a graph structure and the graph is extended by adding a new sub-graph when the agent is in some situation.*

should be linked to. There can be multiple transitions between the existing graph and the new portion.

5.1. Virtual sensors and memory

The events control the action of local behaviors in a situation. That is, depending on what happens in the environment, different events are sent and these change the agent's behaviors. The events are captured by virtual sensors which are attached to the character by a situation. The captured events are stored in a list in character's virtual memory structure. When a situation attaches a sensor, the sensor creates an entry in this list and updates it whenever the character uses the sensor. The contents of virtual memory are wiped out when they are no longer in the situation.

In our system, we use four different sensors to catch four different events.

- **Empty sensor** : This sensor checks for the presence of any agents in a particular position in the environment.
- **Proximity sensor** : This sensor maintains the distance from a character to a particular position in the environment.
- **Signal sensor** : This sensor checks whether or not a signal in the environment is on.
- **Agent sensor** : This sensor checks a particular agent's motion and behavior including position and orientation.

Given events captured from sensors, the event rules inform the agent which behavior functions to evaluate and compose, and can provide arguments to behavior functions that depend on the sensor. For example, suppose a character is in a "crosswalk situation", then the situation attaches a signal sensor to the character to catch the traffic sign. The associated behavior rule checks the sensor and applies a behavior suited either to waiting (if the light says don't walk) or to crossing the street. These dynamic behaviors are composed with others to determine the character's actions. In our system, the event rules are encoded as Python script files and loaded automatically when simulation begins.

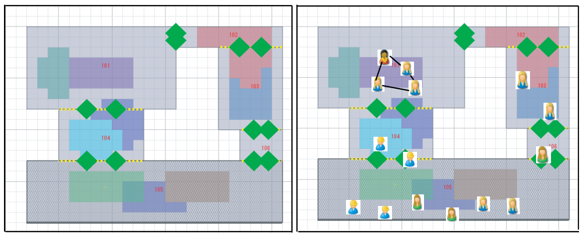

Figure 5: ***Left:*** *Spatial situations can be easily set by drawing directly on the environment. Situation composition can be specified by overlaying regions.* ***Right:*** *Non-spatial situation can be set on the crowd by grouping participants.*

5.2. Situation Composition

In general, agents are under several situations at the same time. This is especially true of non-spatial situations such as group membership that must be composed with behaviors for fixed areas like a bus stop. In our system, the composition of two situations is similar to taking their set union: any state belonging to either is extended onto the graph, and any behavior function required by either is added, as are any sensors.

Some situations must prevent certain behaviors from occurring. For example, in the bench situation, to make an agent sit down at a specific position, the ”don’t turn” default behavior should be ignored in composing the behaviors. This is achieved through event rules that specifically delete the undesirable behavior from the composition.

5.3. Painting interface

The situation-based approach puts an emphasis on environment design because the crowd’s behavior depends on where a particular situation is located (for the case of spatial situation) and what situations the agents are in. For specifying a particular situation on the environment, we adopt a *painting interface*. This allows us to specify situations easily by drawing their regions on the environment directly like drawing a picture on the canvas. Painting is particularly useful for spatial situations. For non-spatial situations, we use standard techniques to select the agents to whom the situation should apply.

The painting interface is based on a layered structure where each layer represents a region for a situation and is saved as a bitmap file. The layered structure makes situation composition easy because it is done by overlapping several layers. Figure 5 shows the painting interface for spatial and non-spatial situations.

6. Experiments and Results

We have performed experiments on a PC with a 1.3GHz Athlon processor and 1GB of memory. At the design stage of a crowd environment, a physical environment and situations that will be fixed for the simulation are defined with the painting interface. At run-time, non-spatial situations and run-time spatial situations are specified. The actual environment file is a Python script including links for situation files which are also encoded as Python scripts. Once all the situations and the environment are set, crowds are created either manually by the user or automatically by the program, and then simulated.

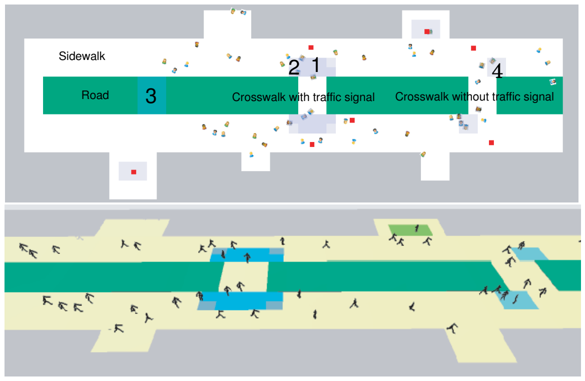
Sidewalk
Road
3
2 1
4
Crosswalk with traffic signal
Crosswalk without traffic signal

Figure 6: ***Top:*** *Plan of the street environment (numbers indicate situations).* ***Bottom:*** *3D view of the street.*

To verify our approach, we tested our system on three different environments, which are the street environment, the theater environment, and the field environment. To clarify the location of situations within an environment, we put an identifying situation number in parenthesis and use the numbers in the associated figures.

6.1. Street Environment

In this environment, we made two crosswalks and two sidewalks on a city street (Figure 6 and Figure 7). One crosswalk has a traffic sign and the other one does not. For the crosswalk with a traffic sign, we composed two situations: the “crossing street” situation (1) and the “traffic sign” situation (2). For the unsigned crosswalk, we just used the unsigned “crossing street” situation (4). Also, in the middle of street away from the two crosswalks, we put an “in-a-hurry” situation (3). At the beginning of simulation, people are walking on the sidewalk. But when they meet the crosswalks, they begin to respond to the situations. At the crosswalk with a traffic sign, they first check the traffic sign using a sensor that is provided by the situation, and wait for the traffic signal before crossing. At the crosswalk without a traffic sign, on the other hand, people just cross the street. Meanwhile, if they are in the “in-a-hurry” situation, they cross the street without using crosswalks. The “in-a-hurry” situation adds running transitions to the agent’s state transition graphs, allowing the crowd to run rather than walk when crossing in an unmarked area.

Due to the sampling strategy, some people who are trying to cross the street might be turning back to the sidewalk at the middle of crosswalk. To solve this problem, we can adjust the weighting constant α in equation 1 for the crossing

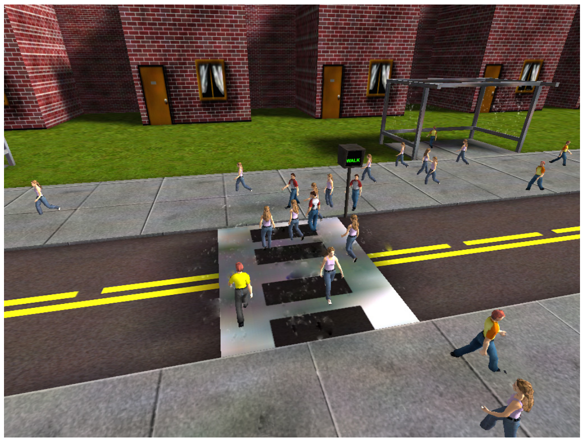

Figure 7: *Crowd behavior in a game engine.*

behavior function so that once they are crossing the street, they continue to walk until they reach the other side of crosswalk.

6.2. Theater Environment

In this experiment, we made a complex theater environment in which there are four different rooms. These rooms are a ticket booth, a lobby, a movie room, and restrooms. In each room, we set several situations and have some of them overlapped and hence composed. Figure 8 shows the theater environment. The various regions are:

- **Ticket booth** : In the ticket booth, we set a "horizontal queue" situation (10) and a "follow" situation (11). When agents enters the "horizontal queue" situation, they stand in a line and try to buy a ticket, one by one. After that, they are guided by the "follow" situation and enter the lobby of the theater.
- **Lobby** : In the lobby, we set two "gathering" situations (5,6), a "stay-in" situation (9), another "horizontal queue" situation (7), a "talk" situation (12) and three "vending machine" situations (8). Among the two gathering situations, one (5) is composed with a "talk" situation (12). For the case of (6), only a "gathering" situation is used. When agents meet the "gathering" situation (6), they gather around for a predefined duration and then spread apart. On the other hand, in the case (5), they gather around for the predefined duration as well, but also sometimes show the talking behavior due to the "talk" situation. At the "vending" machine situation, the crowd stops for a little while, and then moves forward. It is intended to simulate purchasing something from the machine (we were limited by available motion). The "stay-in" situation covers the lobby and restroom and keeps the crowd in those areas before the movie time. This situation adds a signal sensor to all agents in the area. If the signal from the sensor is off, they stay in the lobby or the restrooms. Otherwise, they move to the movie room through path planning.
- **Restroom** : In the restroom, we put three "bench" situations (3), and three "exclusive" situations (2). The "exclusive" situations prevent more than one person from getting into the same restroom at the same time.
- **Movie room** : In the movie room, we set three "seat" situations (1). Each situation makes seven characters sit down in the region, one on a seat. When people are in the situation, they are provided empty sensors and proximity sensors. Using these sensors, they know which seats are empty and which seats are occupied. If they find an empty seat, they move there and sit down.

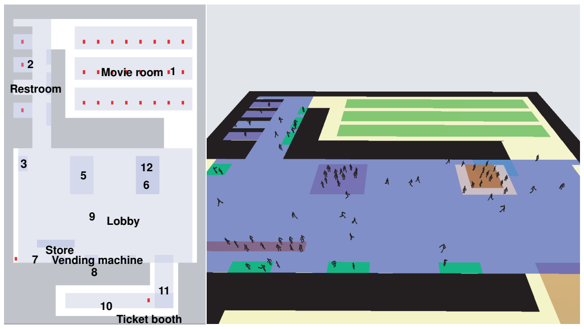

Figure 8: ***Left:*** *Plan of the theater environment(the number indicates situations)* ***Right:*** *3D view of the theater.*

6.3. Field Environment

In order to show that situations can be set at run-time, we create a simple field environment and put a relatively large crowd of 200 agents in the environment (Figure 9). At runtime, a "follow" situation (1) is set by the painting interface. Due to this situation, the crowd flow through the region and gathers at its end. At that point, we set a "group" situation (2) on them (a non-spatial situation). At the same time we disable the "follow" situation, which results in the crowd spreading around. However, if we compose a *close* behavior function with the "group" situation, they gather back.

6.4. Performance measurement

To explore the performance of our approach, we conducted three tests. First, we measured the average memory use of 500 agents for 2,000 simulation steps as the situational complexity of the environment increased. We built up an empty room with the painting interface and put an increasing number of randomly selected situations in the environment. As we increased the number of situations, we computed the memory usage at each of 2000 simulation steps before averaging across steps. To examine the rate of growth of simulation time as a function of situational complexity, we repeated the experiment but this time computed the average amount of time spent on animating the environment (excluding rendering) per frame of motion, or $\frac{1}{30}$th of a second. The results are shown in Figure 10. Note that the complexity grows slowly as a function of the number of situations, suggesting our approach is successful.

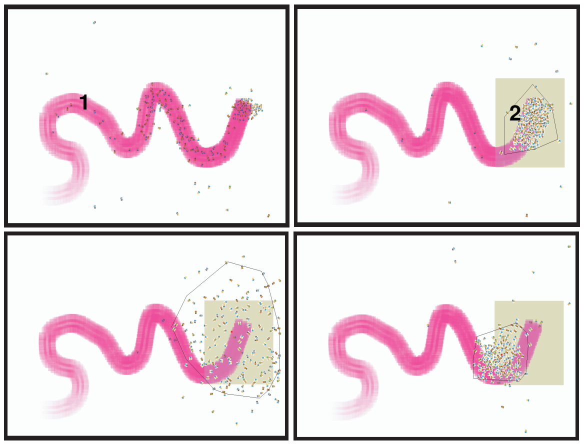

Figure 9: ***Top Left:*** *A crowd is reacting to a "follow" situation.* ***Top Right:*** *We set a "group" situation on the crowd without adding any other behaviors.* ***Bottom left:*** *Due to the group situation, the crowd spreads around (we have disabled the "follow" situation.* ***Bottom right:*** *people gather back when we compose a "close" situation with the group situation.*

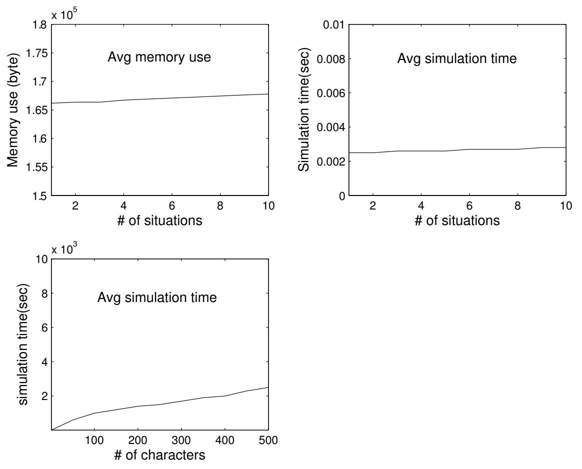

Figure 10: ***Top Right:*** *The average memory use of 500 agent for 2,000 simulation time.* ***Top Left:*** *The average simulation time of 500 agents for 2,000 simulation time.* ***Bottom Left:*** *The average simulation time of crowd with the fixed number of situations.*

We also examined the cost of simulating increasing numbers of agents with a fixed number of situations (10). Again we averaged the time taken to simulate each frame of motion. The results are shown in Figure 10. Even for 500 agents we can compute all the behavioral and motion information in around 2.5ms, representing less than 7.5% of the itme available for each frame.

7. Discussion

In this paper we have introduced a new framework for synthesizing virtual crowds in complex environments. At the high level, we use a situation-based approach that provides a scalable mechanism to control the local behaviors of the crowd. At the low level of the framework, we adopt a probability scheme that composes the influence of several behaviors to drive a realistic motion synthesis system. We have demonstrated that our framework can create complex crowd behaviors through the composition of situations and the composition of behaviors while minimizing data stored in each character.

There are several ways in which we could improve our system. We have not experimented with situations that control the density of the crowd or other multi-agent statistics. This could be done with more intelligent situations that acted as simulation entities in their own right by dynamically adjusting the behaviors they add to agents. From an efficiency standpoint, in our current system we assume that all crowds go through the simulation step at the same time. In order to simulate a massive crowd like 10,000 people, we need to avoid this work in some way, possibly by composing longer pieces of motion that hence require less frequent transitions.

Finally, while our probabilistic composition framework is efficient and works well in the situations we have experimented with, it has limitations as the time scale of actions increases. In other words, the agent needs to remember more and more past states in order to piece together long running actions. We would like to explore other mechanisms for combining behaviors.

Our method is a significant advance in scalable behaviors for characters. Architectures such as ours will be essential for controlling the design complexity of virtual agents as the environments they populate continue to evolve.

Acknowledgments

We thank Rachel Heck, Elizabeth Osten and Eric McDaniel for assisting in video production, proofreading, and other technical matters. We especially thank Aaron San Filippo for visualizing crowds with the *Unreal* game engine. This work was supported in part by NSF grants CCR-9984506 and CCR-0204372 and donations from IBM, Nvidia, Intel, Discreet, Alias/Wavefront and Pixar.

References

[AC01] AYLETT R., CAVAZZA M.: Intelligent virtual environment-a state of the art report. In *Proceedings of Eurographics 2001 STARs* (2001).

[BLA02] BAYAZIT O. B., LIEN J., AMATO N.: Better group behaviors in complex environments using global roadmaps. In *Proceedings of the 2002 Artificial Life (ALIFE)* (Dec. 2002).

[BMdB03] BRAUN A., MUSSE S., DEOLIVERIA L., BODMANN B.: Modeling individual behaviors in crowd simulation. In *Computer Animation and Social Agents(CASA)* (2003).

[EE97] E B., E C.: From crowd simulation to airbag deployment:particle system, a new paradigm of simulation. In *Journal of Electronic imaging* (1997), Annual Conference Series, pp. 94–107.

[FBT99] FARENC N., BOULIC R., THALMAN D.: An informed environment dedicated to the simulaion of virtual humans in urban context. In *Proceedings of EUROGRAPHICS 1999* (1999), Annual Conference Series, pp. 309–318.

[Feu00] FEURTEY F.: Simulating the collision avoidance behavior of pedestrians. In *M.S thesis* (2000), Dept. of EE, the Univ. of Tokyo.

[FMS98] FARENC N., MUSSE S. R., SCHWEISS E.: One step towards virtual human management for urban environment simulation. In *Proceedings of the ECAI Workshop on Intelligent User Interfaces* (1998).

[FMS00] FARENC N., MUSSE S., SCHWEISS E.: A paradigm for controlling virtual humans in urban environment simulations. In *Applied Artificial Intelligence* (2000), vol. 14, pp. 69–91.

[FTT99] FUNGE J., TU X., TERZOPOULOS D.: Cognitive modeling:knowledge, reasoning and planning for intelligent character. In *Proceedings of ACM SIGGRAPH 99* (1999), Annual Conference Series, ACM SIGGRAPH.

[FW01] FORBUS K. D., WRIGHT W.: Some notes on programming objects in *The Sims*, 2001.

[GLM99] GOLDSTEIN S., LARGE E., METAXAS D.: Non-linear dynamical system approach to behavior modeling. In *Applied Artificial Intelligence* (1999), vol. 15, pp. 349–364.

[GSKJ02] GLEICHER M., SHIN H., KOVAR L., JEPSEN A.: Snap-together-motion. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation 2002* (2002), ACM SIGGRAPH.

[HFV00] HELBING D., FARKAS I., VICSEK T.: Simulating dynamics feature of escape panic. In *Nature* (2000), pp. 487–490.

[HM95] HELBING D., MOLNAR P.: Social force model for pedestrian dynamics. In *Physical Review* (May 1995), pp. 4282–4286.

[KG03] KOVAR L., GLEICHER M.: Flexible automatic motion blending with registration curves. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2003* (July 2003).

[KL94] KAVRAKI L., LATOMBE J.-C.: Randomized preprocessing of configuration space for fast path planning. In *IEEE Int. Conf. Robotics and Automation* (1994), pp. 2138–2145.

[Koe] KOEPPEL D.: Massive attack. http://www.popsci.com/popsci/science/article/0,12543,390918-1,00.html.

[KT98] KALLMANN M., THALMANN D.: Modeling objects for interaction tasks. In *Proceeding of Eurographics Workshop on Animation and Simulation 1998* (1998), pp. 73–86.

[LJC01] LI T., JENG Y., CHANG S.: Simulating virtual human crowds with a leader-follower model. In *Proceedings of Computer Animation Conference* (2001), The Computer Graphics Society and the IEEE Computer Society, pp. 93–102.

[MT01] MUSSE S. R., THALMANN D.: Hierarchical model for real time simulation of virtaul human crowds. In *IEEE Transaction on Visualizeaton and Computer Graphics* (2001), pp. 152–164.

[OS94] OVERMARS M., SVESTKA P.: A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics* (1994), pp. 19–37.

[RCB98] ROSE C., COHEN M., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Application 18*, 5 (1998), 32–40.

[Rey87] REYNOLDS C.: Flocks, herds, and schools: A distributed behavior model. In *Proceedings of ACM SIGGRAPH 87* (July 1987), Annual Conference Series, ACM SIGGRAPH.

[TC00] TECCHIA F., CHRYSANTHOU Y.: Real-time rendering of densely populated urban environment. In *Eurographics Rendering* (2000).

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Visualizing crowds in real-time. In *Computer Graphics Forum* (Nov. 2002), vol. 21.

[TLCC01] TECCHIA F., LOSCOS C., CONROY R., CHRYSANTHOU Y.: Agent behavior simulator(abs): A platform for urban behavior develpment. In *Proceedings of GTEC 2001* (2001).

[WCP*] WRAY R., CHONG R., PHILLIPS J., ROGERS S., WALSH. B.: A survey of cognitive and agent architecture. http://ai.eecs.umich.edu/cogarch0/.

[WP01] WATT A., POLICARPO F.: *3D Games:Real time rendering and software Technology*. Addison-Wesley, 2001.