Steering Behaviors for Autonomous Cameras

Quentin Galvane, Marc Christie, Rémi Ronfard, Chen-Kim Lim, Marie-Paule Cani

Steering Behaviors for Autonomous Cameras

Quentin Galvane *
INRIA / LJK, Grenoble, France

Marc Christie
University of Rennes I, France

Rémi Ronfard
INRIA / LJK , Grenoble, France

Chen-Kim Lim
INRIA / Universiti Sains Malaysia

Marie-Paule Cani
INRIA / LJK, Grenoble, France

Figure 1: *Examples of shots captured by our autonomous cameras: (i) the initial interactions between different ethnic groups, (ii) two individual agents are interacting with each other, (iii) the workers are aware of the arrival of the boat, (iv) the workers stop their current interactions and rush to the pier.*

Abstract

The automated computation of appropriate viewpoints in complex 3D scenes is a key problem in a number of computer graphics applications. In particular, crowd simulations create visually complex environments with many simultaneous events for which the computation of relevant viewpoints remains an open issue. In this paper, we propose a system which enables the conveyance of events occurring in complex crowd simulations. The system relies on Reynolds' model of steering behaviors to control and locally coordinate a collection of camera agents similar to a group of reporters. In our approach, camera agents are either in a scouting mode, searching for relevant events to convey, or in a tracking mode following one or more unfolding events. The key benefit, in addition to the simplicity of the steering rules, holds in the capacity of the system to adapt to the evolving complexity of crowd simulations by self-organizing the camera agents to track interesting events.

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

Keywords: camera control, boids, steering behavior

1 Introduction

Crowd simulations generate complex 3D scenes that include both local events (individual actions of crowd members) and larger scale events emerging from many individual behaviors such as a group of individuals simultaneously moving from one location to another. Conveying the resulting animations to spectators is therefore a challenge. Using automatic camera control methods would be a good way to solve the problem. Unfortunately, previous camera control methods focus on maintaining the visibility either of a small number of targets (see [Halper et al. 2001; Oskam et al. 2009; Christie et al. 2012b]), or of large number of targets but from a single viewpoint [Vo et al. 2012]. Other methods are designed for highlighting the motions of isolated characters or of predefined groups of characters [Assa et al. 2008; Lee et al. 2012; Assa et al. 2010], or focus on high-level aspects by encoding elements of cinematographic rules which are only applicable to a small number of characters (see [Lino et al. 2010]). Therefore, capturing a full crowd simulation, with the variety of multi-scale events taking place, calls for the design of new methods, specifically created for crowd simulations.

In this paper, we propose a novel approach for controlling multiple cameras, in the task of conveying as-many-as-possible of the events occurring in a crowd simulation. The original idea is to simulate a team of reporters trying to capture the diversity of behaviors in an environment encompassing both local events (interactions between small numbers of characters) and emerging events that involve many characters. Our approach for animating cameras draws its inspiration from Reynold's steering behaviors [Reynolds 1999]. Indeed, each camera is animated separately as an autonomous agent in our system. This ensures smooth displacements and rotations of cameras. Our camera behaviors are driven by the necessity to both convey local and emerging events, while ensuring a proper distribution of camera agents around the events. More precisely, our camera agents automatically avoid static objects that obstruct the visibility of targeted events. In addition, camera-agent behaviors are specifically designed so that each of them either scouts the environment, searching for new events to shoot, or tracks existing events. While tracking an event, the steering behavior favors good views of the event. When scouting for an event, the steering behavior favors events that are not already followed by other camera agents. We also set up specific interaction rules between camera agents, preventing them from tracking the same event from similar viewpoints. Nearby camera agents are steered away from each other both in position and in orientation. Distant camera agents following the same event are also steered to rotate around the event and take different views of it. All those desirable properties are obtained by allowing the camera agents to be steered horizontally (as in Reynolds' original paper) and also vertically and rotationally (with pan and tilt angles). As a result the user is therefore provided with a gallery of relevant shots on individual and emerging events occurring in crowd simulation, among which the user can select the appropriate animation sequences.

The first key benefit of this approach stands in its simplicity. We show that a small range of simple steering behaviors are sufficient

*e-mail: quentin.galvane@inria.fr

for creating quality camera motions, while covering a wide range of the events occurring in the simulation. Behaviors are crafted in a way which is independent of the crowd simulation model. The second benefit stands in the adaptivity of the method: the same set of steering behaviors enable camera agents to adapt to a wide range of situations, from a small number of cameras in a simulation with many occurring events to a large number of camera agents with a small number of events. In crowd simulations, in which individual and emergent behaviors of the characters are mostly unanticipated, those benefits are especially important.

2 Related work

The computer graphics research community has been showing an increasing interest for techniques to automatically control virtual cameras in 3D environments. The complexity and unpredictable nature of interactive 3D animated scenes, coupled with the necessity to convey contents matching some viewpoint quality metrics, has triggered the study of novel techniques dedicated to virtual camera control.

A first and fundamental requirement of any camera control system is the ability to track (*i.e.* maintain visibility) for one or several targets. Techniques vary according to the knowledge available to the camera control system (known *vs.* unknown environments, known *vs.* unknown trajectories of targets) and to the static or dynamic nature of the 3D scene.

Results in robotics have greatly inspired techniques in virtual camera control. For example, [Oskam et al. 2009] perform a pre-computation of visibility between all possible locations in a static environment, and create a visibility graph that encodes this information. When tracking a single target, the camera relies on the visibility graph to plan the best path towards the target (following different criteria). Changes in environments are partially handled by dynamically updating the visibility graph at the expense of many visibility tests. The approach is limited to a single target.

To avoid the pre-computation of a large visibility graph, [Li and Cheng 2008] compute a local probabilistic roadmap that is defined in the basis of the target (as the target moves, the whole roadmap moves). Camera planning is then performed locally in this roadmap and globally checked against visibility issues, a method that does not require a prior knowledge of the environment.

Indeed, when no prior information on the environment is available to the camera control system, approaches need to reason on local information to recompute viewpoints. Using predicted target movements, [Becker et al. 1997] place the camera so that it would see most of the future target position. [Halper et al. 2001] used a reasoning process on visual properties coupled with hardware projections to track a single target in a reactive way. The tracking of multiple targets has been tackled in [Christie et al. 2012b] by performing a sampling process in the space of camera viewpoints using hardware projections to efficiently compute the visibility of hundred of camera configurations for two or three targets.

All these prior works are restricted by nature to a small number of targets. Recently, [Vo et al. 2012] extended [Becker et al. 1997] to the case of a potentially large group of targets. Their goal was to maximize the visibility of all the targets. In contrast, our goal is different: we want to maximize the visibility of the multi-scale events occurring in the crowd.

In parallel with these techniques that focused on the issue of visibility, other approaches have considered the cinematographic aspect of viewpoint computation (*i.e.* to which degree a shot, a path or a cut satisfies some cinematographic rules and conventions). [Lino et al. 2010], for example, presented a solution for automatically positioning a virtual camera in a 3D environment given the specification of visual properties to be satisfied (on-screen layout of subjects, vantage angles, visibility, scale). It then makes it easy to find the optimal positions for the camera and build a sequence of viewpoints conveying a set of events.

Another interesting feature when selecting a viewpoint is the amount of motion it may convey. Dedicated to the specific task of creating overviews of human motions (*e.g.* from mocap data), [Assa et al. 2008] cast the problem of camera control as an energy minimization process guided by the conjunction of external forces (describing the viewpoint quality at each location and time) and internal forces (enforcing smoothness on the generated path). As a result, the system generates a sequence of camera paths (with edits) using an optimization process on a potential field defined by the aggregation of forces. While purely based on character motion analysis and viewpoint quality heuristics, the process generates meaningful overviews of human motions without integrating semantic aspects. The approach was extended to multiple characters [Assa et al. 2010], to convey motions of multiple humans in a real-time context. The authors consider a set of virtual cameras animated in real-time that shoot multi-character motions. Starting with random camera motions or from simple heuristic motion rules, the system enables to select viewpoints that maximize a correlation measure between the motion of the characters in the scene, and the projected motion of the characters on the screen (a strong correlation being a good indicator of viewpoint quality). In contrast to our work, this method offers no means of controlling a large set of virtual cameras in a complex environment.

Closer to our work, [Lee et al. 2012] proposed to automatically extract *social events* from the joint analysis of multiple character motions, e.g. related to trajectories and similarities in distance and direction of character motions. Events are then processed, analyzed for spatio-temporal correlation and ranked so as to generate motion clip segmentation. The motion clip segmentation is used as a basis to express an optimization problem on the camera parameters for each motion clip (actually long motion clips can be separated into segments solved individually). Following the lines of [Assa et al. 2008], the optimization process is guided by internal, external and continuity forces shaping a potential field. Continuity forces integrate the 180-degree rule as well as the jump-cut rule. The computational cost of the optimization process remains significant (1 to 3 minutes).

None of these contributions directly addressed the problem of the real-time control of multiple cameras, self-organizing to cover the largest possible part of a crowd simulation scene and for maximizing the coverage of on-going events. This is the problem we are tackling here.

3 Background on steering behaviors

In this section, we review some important concepts related to steering behaviors. Some of them will be extended to the case of steering cameras in Section 4.

3.1 Agent dynamics

In Reynolds' approach, autonomous agents are driven by steering forces [Reynolds 1999] and their position is updated at every time step in the simulation as follows:

Algorithm 1 Agent dynamics. At each time step, the acceleration of an agent i at time t (denoted $\gamma_i(t)$) is computed as a sum of forces, and a Euler integration is performed to compute velocity $v_i(t)$ and position $p_i(t)$.

$t = 0$
while simulation is running **do**
  for all agents i **do**
    Update steering forces $F_{ij}(t)$
    $\gamma_i(t) = \sum_j F_{ij}(t)$
    $v_i(t) = v_i(t - \delta t) + \gamma_i(t)\delta t$
    $p_i(t) = p_i(t - \delta t) + v_i(t)\delta t$
  end for
  $t = t + \delta t$
end while

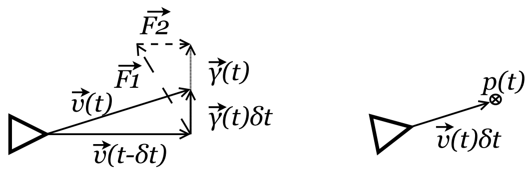

Figure 2: *Computation of forces applied to an agent. The agent is represented as a triangle: (a) the agent's velocity $v(t)$ is updated by integrating acceleration expressed as a sum of forces; (b) the agent is then oriented along the newly computed velocity $v(t)$.*

3.2 Steering forces

Simple steering forces enable the creation of complex simulations with emerging behaviors. A simulation is created using a set of agents that are defined by a set of characteristics: position, velocity, radius, mass, maximum force and maximum speed. Motions of the agents are generated using these simple characteristics, and by applying a simple particle based algorithm where each particle is represented as a "vehicle". In order to improve the efficiency of the entire computation and implementation of the system, Reynolds specified a simple but efficient constraint: agents always move in their forward direction. Despite this simplifying assumption, Reynolds was able to present a rich vocabulary of steering behaviors driven by specific steering forces including seeking, fleeing, pursuit, evasion, offset pursuit, arrival, obstacle avoidance, wandering, path following, wall following, containment, flow field following, unaligned collision avoidance, separation, cohesion, alignment, flocking and leader following.

Many of these different forces are based on a simple concept: the force that will be applied to the agent is computed by subtracting the desired velocity related to the behavior with the current velocity of the agent. The acceleration being the sum of all the forces, at each time step, the velocity will be updated by adding part of the acceleration (depending on the time step) which will reduce the difference between the velocity and the desired velocity. The seeking force devised by Reynolds and illustrated in Figure 3(a) is based on this concept. The desired velocity is computed by first finding the direction vector, and then multiplying it by the maximum speed of the agent. The direction of the desired velocity is the vector between the target and the agent.

For the sake of clarity, we hereby illustrate the design of the arrival behavior: an agent should slow down when reaching its target. When the distance between the agent and the target becomes smaller than a threshold distance, the desired velocity needs to decrease with the distance. Therefore, it is multiplied by the by the division of the distance and the threshold distance. When the distance is greater than the threshold distance, the normal seeking force is applied. Figure 3(b) illustrates this arrival behavior.

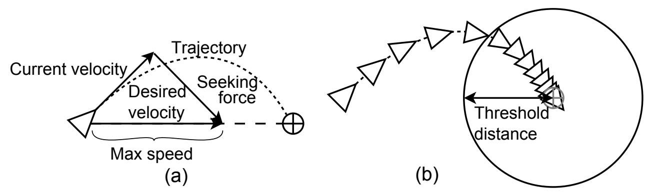

Figure 3: *Illustration of two behaviors: (a) seeking behavior, (b) arrival behavior*

4 Steering cameras

In this section, our objective is to implement appropriate steering behaviors for our camera agents. The task assigned to camera agents is to convey individual and emerging events occurring in a crowd simulation. For this purpose, we propose two steering behaviors:

- **Scouting**: default behavior when a camera is not following an event; the camera is searching the scene for events to track.
- **Tracking**: behavior of a camera while it is following an event; the camera is in recording mode during tracking.

In both cases, we need to extend the dynamics of Reynolds' agents to desynchronize the agents' direction of motion from their orientation (a strong hypothesis proposed by Reynolds). We will explain how this extension is performed in 3D with cameras that can change their pan and tilt angles independently of their position and motion direction, while simultaneously changing their elevation. We will then describe camera-specific forces and torques, whose combined effects generate the desired camera behaviors during scouting and tracking.

4.1 Targets and events

We define an *event* in the simulation as a spatio-temporal segment where a particular behavior is observed, should it be relative to an individual behavior, a one-to-one interaction behavior or a group behavior (*e.g.* flocking, herding, and leader-following, where an entire group of characters follows a recognizable and coherent motion pattern). Each event involves a number of characters (characters concerned by this event in the crowd simulation). Our representation is detached from any specific crowd simulation model but requires a step to extract the events from the simulation using geometrical features relative to the characters.

More precisely, the computation of events is based on the following information:

- position and speed vector of the characters;
- orientation of the characters (forward vector).
- type of the characters (if the simulation uses several types of characters);

In this work, we propose to extract two types of events:

- *one-to-one interaction*: two characters face each other for a long time without moving (see Figure 4(a));

- *group motion*: emergent behavior observed when a group of characters is moving together in the same direction (see Figure 4(b)).

(a) (b)

Figure 4: *During scouting, cameras search for interesting events: (a) One-to-one interaction event; (b) Crowd behavior event.*

Our *one-to-one interaction* events are easily detected when pairs of characters face each other with zero velocity. Our *group motion* events are detected by studying a group of characters, based on their direction, speed, density and homogeneity in the following way:

- Average direction: by summing the normalized speed vectors of each character and then dividing the total by the number of characters in the group, we get a vector which magnitude will be related to the alignment. Thus the length of this vector gives us a good estimate of the alignment of the characters in the group: the closer the value is to one, the more the group is aligned.
- Average speed: The speed of each character is closer to the average speed.
- Density: the amount of characters per unit of surface.
- Homogeneity: the number of characters in the group belonging to the same type.

In order to efficiently implement the detection of group motions, we rely on a quad-tree representation. The scene is divided into a 2D multi-resolution grid and, starting from the finest (highest) resolution, we evaluate in each cell whether the characters in the cell form a group motion. At the next (coarser) level, we create a group containing all cells with similar group motions and repeat the process until it reaches the coarsest level in the hierarchy. During group motion, both the group and its motion are continuously updated over time. New group motion events are created when a new group is detected and deleted when the group becomes too small.

4.2 Camera dynamics

In Reynolds paper, one simple assumption is made: the agents are always moving in the direction or their orientation, *i.e.* their velocity vector (that gives the movement direction and the speed) and their forward vector (gives the orientation of the agent) are aligned. Implementing steering behaviors for a camera agent however requires to distinguish the camera orientation from its direction, typically moving in one direction while filming in another one.

Simple strategies that consist in re-orienting the camera at each time step towards the center point of the group of characters involved in an event would fail. The varying number of characters in an event would make the center move significantly at each time step, leading to undesirable jerky motion. Moreover, no targets are available in the scouting behavior (since no events are detected). Instead, we propose to apply a dynamic model on the camera orientation, expressed as a torque. Camera position and orientation are therefore computed separately: steering forces are applied to move the camera around the scene and rotation forces are applied to turn the camera around. The model intrinsically produces smoother changes in orientation.

Figure 5 illustrates the computation of a new camera orientation given different forces (in 2D for illustration). First, the angular acceleration is computed using the torques. Then, the rotational velocity is updated using the acceleration. Finally, the new orientation is computed with the angular velocity. A camera agent is represented by its 3D position in the Cartesian space p_i, a pan angle θ_i and a tilt angle ϕ_i. For the sake of clarity, we denote the camera orientation as a quaternion q_i and rewrite the camera dynamics equations using quaternion multiplication. The algorithm 2 details the computation of the new orientation used at each time step.

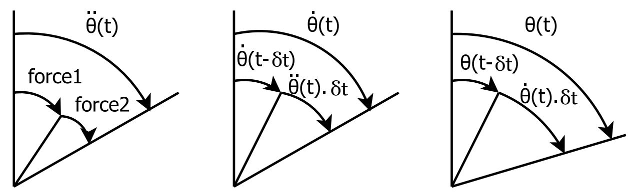

Figure 5: *2D representation for the computation of the camera's horizontal orientation (θ) from the angular acceleration ($\ddot{\theta}$) and the angular velocity($\dot{\theta}$).*

Algorithm 2 Camera dynamics: $\ddot{q}_i(t)$ represents the rotation acceleration of agent i at time t, $\dot{q}_i(t)$ the rotation velocity and $q_i(t)$ the orientation.

while simulation is running **do**
  for each camera agent i with orientation q_i **do**
    Update steering forces $F_{ij}(t)$ and torques $T_{ij}(t)$ of agent i

    Update camera position as usual
    $\ddot{q}_i(t) = \sum_j T_{ij}(t)$
    $\dot{q}_i(t) = \dot{q}_i(t-\delta t)\ddot{q}_i(t)\delta t$
    $q_i(t) = q_i(t-\delta t)\dot{q}_i(t)\delta t$
  end for
  $t = t + \delta t$
end while

4.3 Camera steering forces

We now detail the design of steering forces associated to each camera. The essential requirements are: maintaining a good framing of an event, maintaining visibility of an event, keeping a given distance to an event and finally ensuring that different cameras cover different viewpoints of the same event.

In order to control the camera position, five forces have been designed: target following, obstacle avoidance, camera separation, wandering and containment. We now review them one by one.

Framing force: moves the camera towards the closest optimal camera position in terms of framing (composition of characters on screen (without considering obstacles). The goal of this force is to ensure that the camera maintains a specific framing of an event, *i.e.* that characters composing the event stay within the camera frustum. To ensure this framing, we designed a force which attracts the camera agent towards a range of viewpoints defined by two points A and B. This range represents a continuous set of viewpoints (see examples in Figure 7) for which the framing can be easily computed. The computation relies on Lino and Christie's frame composition technique [Lino and Christie 2012] by considering that a group of characters can be framed by framing their left-most and right-most characters on the screen as illustrated in Figure 6(a).

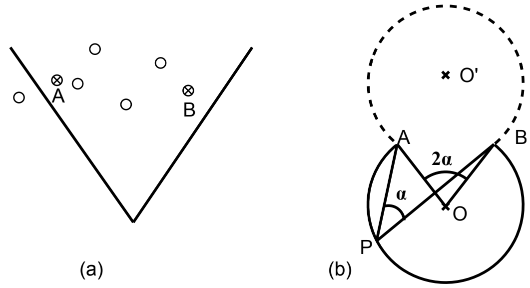

Figure 6: *Framing force: (a) selecting the left-most and right-most characters of a group* w.r.t *the current camera position enables (b) the construction of a specific range of viewpoints, for which the framing of both characters is ensured.*

Indeed, for two points A and B, viewed with a constant angle α, the camera must lay on a circle of radius 2α [Lino and Christie 2012]. Figure 6(b) shows the two arcs representing the possible positions of the camera. In our case, we choose the angle α as a percentage of the field of view and then compute the two arcs.

However, although all cameras positioned on these two arcs maintain a constant viewing angle, the viewpoints which are too close to A or B (displayed in red dashed lines in Figure 7) are not suitable. Indeed, when framing a group of two characters we prefer to avoid these less informative shots (very close shots to one or the other agents). To account for this problem, we propose to modify Lino and Christie's approach by changing the range of possible camera positions when the angle $(\vec{O'P}, \vec{O'O})$ is less than a minimum value (β), meaning the camera is too close to the agents (see Figure 7). We rely on a Bezier curve to define this range. The three control points P0, P1 and P2 are computed by using the values of β, and P1 is the intersection of the tangent lines to the two circles in P0 and P2.

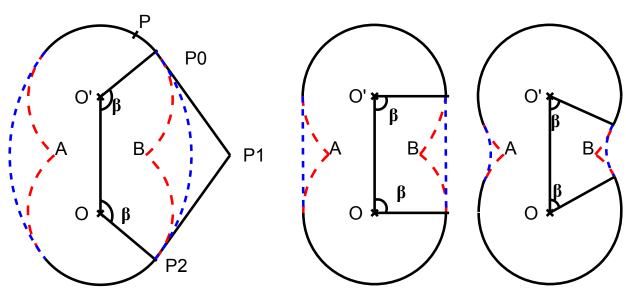

Figure 7: *Framing force: modifying the range of viewpoints around two characters A and B, in function of β. The framing force pushes the camera to the closest point on this range.*

The force is then easily expressed by using Reynolds' arrival force targeting the closest point from the camera agent to the range of possible viewpoints. In addition, when the camera is filming a large group, a force elevating the camera is added to mimic crane cameras and to improve the visibility of the agents.

Obstacle avoidance force: moves the camera agent on one side or the other depending on the position of the obstacle in the frustum. This force differs from the obstacle avoidance force presented in Reynolds' paper: the process loops on the list of obstacles entirely or partially included in the frustum. Each time an obstacle at position o is in the frustum and is closer to a camera agent position p_i than a target position k is, we consider the obstacle may potentially occlude the target. To prevent these potential occlusions, a force is applied to the camera depending on whether the obstacle is in the left part of the frustum (in such case the force moves the camera on the right) or in the right part of the frustum (in such case the force moves the camera on the left). The position of the target k is computed as the center of all characters involved in the event. For each obstacle, the force is computed by subtracting the current velocity to a desired velocity. See Algorithm 3 for details.

Algorithm 3 Obstacle avoidance: computes a sum of forces F_{obs} that pushes the camera on the left or the right according to the relative positions of the obstacles and the target. l_i represents the normalized look at vector (orientation) of camera agent i at time t, r_i represents the normalized right vector of camera agent i at time t and v_{max} is the maximum allowed velocity for the camera.

```
for each obstacle at a position o in the frustum of camera agent i do
    // check whether the obstacle is closer to the
    // camera than the target is
    if (o − p_i) · l_i < (p_i − k) · l_i then
        // if obstacle is on the right hand side of the camera
        if (o − p_i) · r_i > 0 then
            u = −r_i v_max // compute a desired velocity to the left
        else
            u = r_i v_max // compute a desired velocity to the right
        end if
        // subtract the current velocity to the desired force
        F_obs = F_obs + (u − v_i)
    end if
end for
```

Camera separation force: moves a camera agent away from other camera agents that are too close and are looking in the same direction. In scouting mode, this force ensures a degree of diversity by separating similar cameras and locally improving the coverage of different events. In tracking mode, this same force enables the cameras to pick different views of the same event (useful when tracking a large group). The corresponding force is computed as described in Algorithm 4: if the distance between a camera agent c and a camera agent $i \neq c$ is lower than a threshold d_{max}, a force is applied either towards the right or the left of the camera. The intensity of this force is proportional to the angle between the look at vector l_i of camera i and the look at vector l_c of camera c, computed with $max(0, l_c \cdot l_i)$.

Wandering force: moves the camera in the scene while searching for new events in scouting mode. This force is computed by updating a wandering direction at each time step and computing a desired velocity from this direction (see [Reynolds 1999] for more details).

Containing force: prevents the camera from wandering away in the scene. When the camera agent wander too far from the crowd, a steering force will push it back toward the center of the crowd (see [Reynolds 1999] for more details)

4.4 Camera steering torques

For the camera rotation, three torques were designed: aiming, camera avoidance and wandering.

Algorithm 4 Camera separation: computes the sum F_{sep} of forces that pushes the camera c away from other camera agents. p_i and p_c represent the positions of camera agents i and c and r_c is the right vector of camera c.

for each camera i different from camera c **do**
  if $|p_i - p_c| < d_{max}$ **then**
    // move the camera to the left or to the right
    if $(p_i - p_c) \cdot r_c > 0$ **then**
      $u = -v_{max} r_c$ *// compute a desired velocity to the left*
    else
      $u = v_{max} r_c$ *// compute a desired velocity to the right*
    end if
    // subtract the current velocity to the desired velocity
    // and scale it
    $F_{sep} = F_{sep} + max(0, l_c \cdot l_i)(u - v_c)$
  end if
end for

Aiming torque: rotates the camera towards the "optimal" orientation. This torque is inspired by Reynolds' arrival force (see Figure 3) and transposed to the problem of camera orientation. Given a desired camera orientation q_d, and the orientation q_i of a camera agent i, we need to compute the appropriate torque to reach the target camera orientation within the limits of a maximal rotational speed α_{max}. We first compute the difference between quaternions q_d and q_i to extract the angle q_α and axis q_a of rotation.

As defined in the arrival behavior (see Figure 3), if the angle α is below a given threshold α_t (meaning the angle is getting close to its desired value q_d), we progressively reduce the rotational speed. If the angle α is above the threshold, the rotational speed is set to a maximum value α_{max} at each time step. Algorithm 5 presents the computation of the torque T_{aim} depending on the angle α and the threshold angle α_t.

Algorithm 5 Aiming torque: computes a torque T_{aim} applied to camera c. The rotational velocity of camera c is denoted by $\dot{q}_c$.

// computing the quaternion difference between q_d and q_c
$q = q_d . q_c^{-1}$
// q^α is the rotation angle of quaternion q
if $q^\alpha > \alpha_t$ **then**
  $\alpha = \alpha_{max}$
else
  $\alpha = \alpha_{max} q^\alpha / \alpha_t$
end if
// compute a rotation of angle α and axis q^a
$T_{aim} = (\alpha, q^a)\dot{q}_c^{-1}$

Figure 8 illustrates how the magnitude of the force (*i.e.* the value of the angle) is computed.

Camera avoidance torque: turns cameras away from each other when they share part of their frustum.

The computation of this torque is two-fold. For each pair of cameras, we first check whether their frustums intersect. This collision test is simplified by using a 2D representation of the frustum. The intersection of the left and right axis (representing the frustum side planes) of the cameras are computed. Then the collision is detected if, for at least one of these four intersection points, their projections on the the forward vectors of the two cameras are contained inside the frustum of the respective cameras (i.e. the distance from the projection point to the camera origin is greater than the near plane distance of the frustum, and smaller than the far plane distance).

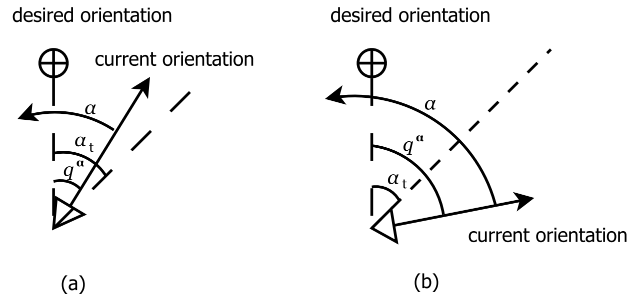

Figure 8: *Aiming torque inspired from the arrival behavior: two cases (a) q^α is less than α_t and (b) q^α is greater or equal to α_t.*

Figure 9(a) shows an example where no collision are detected: the two projections P_j of each intersection point I_i on the forward vectors of the cameras are never inside both of the frustums. Conversely, Figure 9(b) shows an example of frustum intersection: the two projections $P1$ and $P2$ of the intersection point $I1$ are respectively inside the frustum of the camera 1 and 2. If a frustum collision is detected, the desired rotation velocity ($\dot{q_d}$) will be oriented depending on the relative positions of the two cameras. The process is detailed in Algorithm 6.

Algorithm 6 Camera avoidance torque: computes a torque T_{avoid} for a camera c. The rotational velocity of camera c is denoted $\dot{q_c}$ and the right vector of camera c is denoted r_c. In addition, α_{vmax} represents the maximum rotational speed and u_c the up vector of the camera.

for all cameras i different from current camera c **do**
  if frustum of i intersects frustum of c **then**
    if $(p_i - p_c) \cdot r_c > 0$ **then**
      $\dot{q_d} = (-\alpha_{vmax}, u_c)$
    else
      $\dot{q_d} = (\alpha_{vmax}, u_c)$
    end if
    $T_{avoid} = T_{avoid} \cdot \dot{q_d} \cdot \dot{q_c}^{-1}$
  end if
end for

Camera wandering torque: used for scouting new events. This torque is created by computing a wandering direction and aiming at a point along this direction.

5 Experimental results

For testing our approach, we implemented a realistic crowd simulation framework, based on a large number of characters (over a hundred) belonging to different ethnic groups. The task of the camera agents is to report as many events as possible, and to record or broadcast the best possible coverage of those events.

5.1 Crowd simulation

Our crowd simulation is actually based on Reynolds steering behaviors(see [Lim et al. 2013]). The scene simulates two distinct types of ethnic groups in Weld Quay, Malaysia back in the 19th century. The interactions of these two ethnic groups are highly influenced by their standard roles in the trading port. The main roles of each of these ethnic groups are as follows:

- Malay: local inhabitant/ seller at the market place

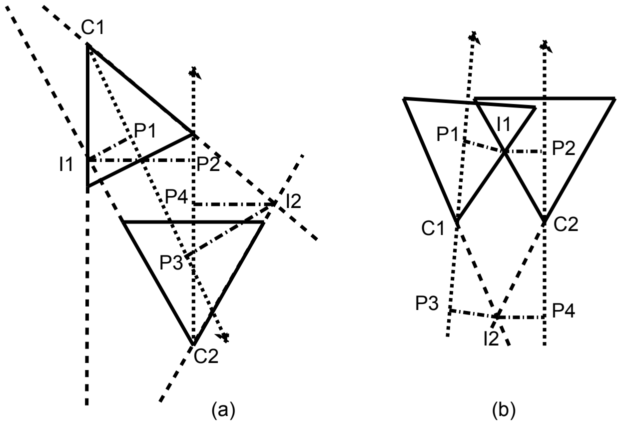

Figure 9: *Frustum collision test; the continuous lines indicate the frustum and the dotted arrows indicate the forward vectors.*

- Indian: imported worker

There are various interactions either between individual agents and within the same ethnic groups or among other different ethnic groups that are transpired in the trading port and we experiment our camera steering behaviors based on two scenarios as follows:

- Residents: Initialized interactions with any agents from the other ethnic group to sell their local products. These events are the one-to-one interactions that we are interested to identify.
- Workers: Unload the goods at the pier and download the goods at the containers. Emergents group movement should result from this shared goal.

The simulation takes place in a moderately complex environment (Figure 10) with the following scenario. At initialization, the workers are wandering in the market, the residents are trying to sell their products. When the boat arrives, some of the workers will try to reach the pier to unload the goods. When the boat is empty, it sails away and the workers can go back to the market place.

Figure 10: *Virtual environment of Weld Quay simulating the interactions between two ethnic groups in Malaysia back in the 19th century.*

5.2 Implementation details

For the purpose of efficiency, we used a quad-tree to accelerate crowd simulation and event detection. As a result, each camera agent is aware of all events and characters within its field of view. Figure 11 shows the 2D spatial layout of characters (red and blue dots) in an overview of our virtual environment.

Figure 11: *Using a quad-tree representation to reduce computational cost in querying characters in camera frustums and detecting events.*

5.3 Qualitative evaluation

To evaluate our approach, we propose an *activity* metric that shows the overall presence of *active* characters (characters involved in an event) compared to *inactive* characters (characters not involved in an event). The activity metric is composed of a score representing the active characters and a score representing the inactive characters. These scores are evaluated for each camera by integrating a weight on characters: the closer a character is to the camera, the more he will affect the overall score of the camera.

Algorithm 7 Activity metric

```
for all cameras do
    for all characters seen by the camera do
        if character is Active then
            scoreActive += 1/distance(camera, character)
        else
            scoreIdle += 1/distance(camera, character)
        end if
    end for
end for
```

We compared the activity metrics for our system with a much simpler solution with static cameras placed at strategic positions (to ensure that they would not lose sight of the crowd – information that our autonomous camera agents do not have). Figure 12(a) shows the temporal evolution of the ratio of active *vs.* idle characters captured by autonomous cameras. Figure 12(b) shows the results for static cameras (dark gray values correspond to active characters). We can see that, even though it is less stable (due to the scouting state) the average proportions of active characters remains more important than with statics cameras. Indeed, autonomous cameras try to get closer to the active characters, and thus improve scores whereas static camera are only waiting for active characters to pass by. We can observe the same phenomenon with eight static and moving cameras in Figure 13. Moreover, autonomous cameras are able to maintain a good framing of their targets while performing elaborate and dramatic camera motion, as can be seen in the accompanying video. Figures 4(a) and 4(b) display typical shots for one-to-one interactions and group motion events.

In Figures 12 and 13 we can observe some fluctuations in the graph.

These fluctuations are due to the fact that when an event ends, it might not have any other events in its field of view since the camera was focusing on this specific event, and thus it sometimes results in a drastic drop in the ratio idle/active while the camera is searching for a new event.

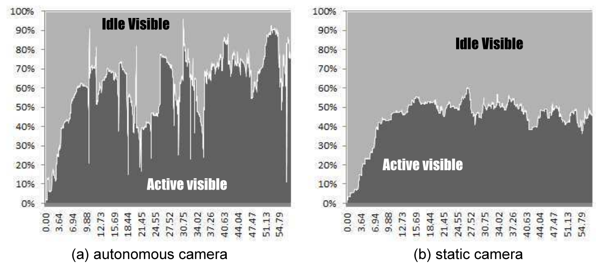

Figure 12: *Ratio between the active score and the idle score for groups of 4 autonomous (a) and static (b) cameras*

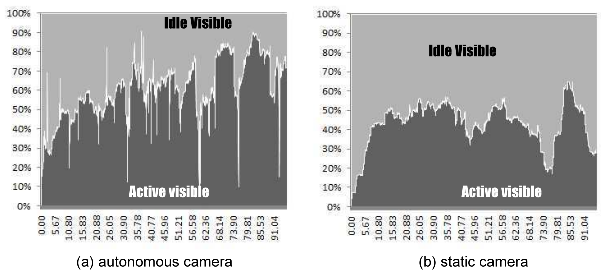

Figure 13: *Ratio between the active score and the idle score for groups of 8 autonomous (a) and static (b) cameras*

5.4 Quantitative evaluation

One way to present quantitative results is to express the number of active characters that are being viewed by the cameras, in comparison with the number of active characters not viewed by the camera. Results are reported in Figure 14 with 4 autonomous cameras and Figure 15 with 8 autonomous cameras. These results illustrate both the capacity of a small number of cameras to cover a crowd simulation (in comparison to 4 static cameras), and show that an increase in the number of cameras improves the results essentially for autonomous cameras.

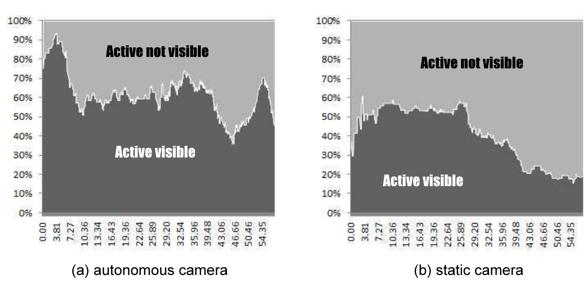

Figure 14: *Temporal evolution of the ratio between active characters viewed by the cameras and active characters not viewed by the camera, considering 4 autonomous camera agents (a) and 4 static cameras (b).*

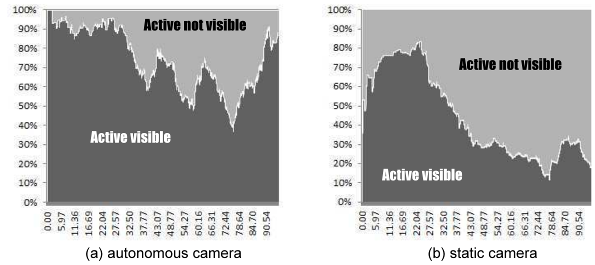

Figure 15: *Temporal evolution of the ratio between active characters viewed by the cameras and active characters not viewed by the camera, considering 8 autonomous camera agents (a) and 8 static cameras (b).*

In terms of performance, the method remains efficient and can steer 30 camera agents in real-time (15fps) on a Core i7@2.4GHz running Unity 4 with 100 virtual characters. The bottleneck is essentially due to the number of virtual characters to simulate. The drop in framerate comes from the cross-computations between all camera agents necessary in the camera avoidance and camera separation forces.

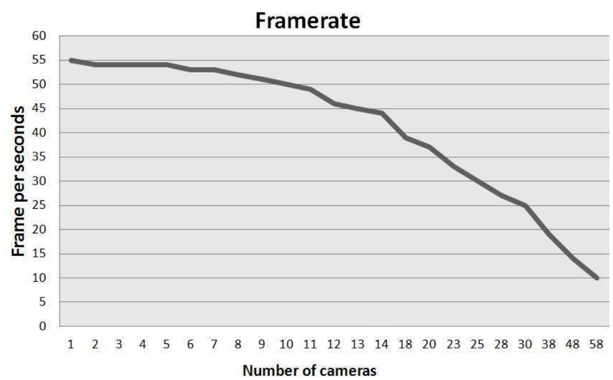

Figure 16: *Performance our crowd simulation when increasing the number of camera agents.*

6 Limitations and future work

Currently our model is limited to two event classes - one-to-one interactions and group motions. Future work will investigate a richer model of event categories [Lim et al. 2013]. Extending the method to other event types may require additional rules with regards to social behaviors [Epstein and Axtell 1996]. For instance, wall following, leader following, evading and so on might be useful in capturing scenarios such as queuing to exceed narrow path, robbery and fighting scenes in a more complex way.

The focus in this paper has been on the simple task of tracking as many events as possible. In a more realistic scenario, we should take into account that some events may be more important than other. For instance, repetitive events may not require as much coverage as unique events. Simple strategies may be used to focus more on the first instance of each event class. Similarly, events with a long duration may not require a full coverage. More work is needed to work out strategies that can reason about event classes and decide between competing events.

Another limitation of our current implementation is that we use a single tracking mode. In future work, other camera behaviors could easily be added as specialized idioms, in the fashion of [He et al. 1996]. For instance, a pair of cameras shooting a group both from the front and the back provides a better coverage of the event. As another example, with two cameras covering a single event, it becomes possible to come closer to the event and even to shoot from within the event (in media re). Other idioms that are yet to implement, but would be easy to do, are stopping the camera to let the group move on-screen; cross-cutting between forward and backward shots; cross-cutting between close-ups.

At this point, our model also does not take into account rules of editing [Ronfard 2012]. In future work, we would like to include them using direct communication between cameras, so that the coverage of all cameras can be easily edited together [Lino et al. 2011; Christie et al. 2012a]. Examples of editing rules that can easily be taken into account by two-way communication between cameras are the 30-degree rule (cameras viewing the same events should be separated by at least thirty degrees of rotation around the event) and the 180-degree rule (cameras viewing the same event should stay on the same side of the event).

We also would like to extend our framework to the case of coordinated cameras, where camera behaviors can be chosen by a "director" agent, taking higher level goals into account, as in the live broadcast of an event.

7 Conclusion

We have proposed an original method for automatically computing shots of crowd simulations, using a predefined number of autonomous cameras. Our autonomous cameras react to crowd animation events using specialized camera steering behaviors and forces based of Reynolds' model. By separately designing forces applied to the camera position, and torques applied to the camera orientation, our method offers a fine control over the camera agents. Experimental results show that the method provides a good coverage of events in moderately complex crowds simulations, with consistently correct image composition and event visibility. Overall, the strength of this approach lies in its simplicity and its ability to be extended with new steering behaviours.

Acknowledgements

Part of this work was funded by the ANR project CHROME ANR-12-CORD-0013 and by the ERC advanced grant EXPRESSIVE.

References

ASSA, J., COHEN-OR, D., YEH, I.-C., AND LEE, T.-Y. 2008. Motion overview of human actions. In *ACM SIGGRAPH Asia*, 115:1–115:10.

ASSA, J., WOLF, L., AND COHEN-OR, D. 2010. The virtual director: a correlation-based online viewing of human motion. *Computer Graphics Forum 29*, 2, 595–604.

BECKER, C., GONZLEZ-BAOS, H., LATOMBE, J., AND TOMASI, C. 1997. An intelligent observer. In *Experimental Robotics IV*, Springer Berlin Heidelberg, O. Khatib and J. Salisbury, Eds., vol. 223 of *Lecture Notes in Control and Information Sciences*, 151–160.

CHRISTIE, M., LINO, C., AND RONFARD, R. 2012. Film editing for third person games and machinima. In *Workshop on Intelligent Cinematography and Editing*.

CHRISTIE, M., NORMAND, J.-M., AND OLIVIER, P. 2012. Occlusion-free camera control for multiple targets. In *Symposium on Computer Animation*, 59–64.

EPSTEIN, J. M., AND AXTELL, R. 1996. *Growing artificial societies: social science from the bottom up*. The Brookings Institution, Washington, DC, USA.

HALPER, N., HELBING, R., AND STROTHOTTE, T. 2001. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum 20*, 3, 174–183.

HE, L.-W., COHEN, M. F., AND SALESIN, D. H. 1996. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *ACM SIGGRAPH*, 217–224.

LEE, T.-Y., LIN, W.-C., YEH, I.-C., HAN, H.-J., LEE, J., AND KIM, M. 2012. Social-event-driven camera control for multicharacter animations. *IEEE Transactions on Visualization and Computer Graphics 18*, 9, 1496–1510.

LI, T.-Y., AND CHENG, C.-C. 2008. Real-time camera planning for navigation in virtual environments. In *Proceedings of the 9th international symposium on Smart Graphics*, Springer-Verlag, Berlin, Heidelberg, SG '08, 118–129.

LIM, C. K., CANI, M.-P., GALVANE, Q., PETTRE, J., AND TALIB, A. Z. 2013. Simulation of past life: Controlling agent behaviors from the interactions between ethnic groups. In *Digital Heritage International Congress*, (to appear).

LINO, C., AND CHRISTIE, M. 2012. Efficient Composition for Virtual Camera Control. In *ACM Siggraph / Eurographics Symposium on Computer Animation*, P. Kry and J. Lee, Eds.

LINO, C., CHRISTIE, M., LAMARCHE, F., SCHOFIELD, G., AND OLIVIER, P. 2010. A real-time cinematography system for interactive 3d environments. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '10, 139–148.

LINO, C., CHOLLET, M., CHRISTIE, M., AND RONFARD, R. 2011. Computational model of film editing for interactive storytelling. *International Conference on Interactive Digital Storytelling Lecture Notes in Computer Science 1*, 2.

OSKAM, T., SUMNER, R. W., THUEREY, N., AND GROSS, M. 2009. Visibility transition planning for dynamic camera control. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '09, 55–65.

REYNOLDS, C. 1999. Steering behaviors for autonomous characters. In *Game Developers Conference*, 763–782.

RONFARD, R. 2012. A Review of Film Editing Techniques for Digital Games. In *Workshop on Intelligent Cinematography and Editing*, ACM, Raleigh, États-Unis, R. M. Y. Arnav Jhala, Ed.

VO, C., MCKAY, S., GARG, N., AND LIEN, J.-M. 2012. Following a group of targets in large environments. In *Motion in Games*, Springer, M. Kallmann and K. E. Bekris, Eds., vol. 7660 of *Lecture Notes in Computer Science*, 19–30.