Optimization-based computation of locomotion trajectories for Crowd Patches

Authors*
Institution

Abstract

Abstract, abstract, abstract ...

CR Categories:

Keywords: keywords go here

1 Introduction

Video Games are constantly displaying larger and more lively virtual environments due to increased computational power and advanced rendering techniques. For example, the recent Grand Theft Auto (GTA) game [Rockstar-Games 2004] takes place in Los Santos and its surroundings, a completely virtual city. In spite of the impressive quality and liveliness of the scene, Los Santos still remain relatively sparsely populated with virtual people. The reason for this phenomenon is the large computational cost required to get an *ambient crowd* in such large environments. To address this issue, the technique of *Crowd Patches* has been recently introduced in [Yersin et al. 2009].

Crowd patches are precomputed elements (patches) of crowd animations. Patches are time-periodic to be endlessly played in time. The boundary conditions of precomputed animations are accurately controlled to enable combining patches in space (i.e., characters can move from in-between patches), and compose large ambient crowds. This technique eases the process of designing an ambient crowd in an efficient way.

One problem with this technique however, is to compute internal animation trajectories for patches, that satisfy both, time-periodicity and boundary conditions among patches. Satisfying both of these constraints is difficult, because it is equivalent to computing collision-free trajectories that exactly pass through spatio-temporal waypoints (i.e., at some exact position in time) whilst at the same time solving possibly complex situations of interactions with other agents (collision-avoidance). In addition, trajectories should look as natural as possible.

In this paper we propose a new optimization-based method to compute these internal trajectories. Our method starts by assigning linear trajectories which are easy to compute and satisfy both, periodicity and boundary conditions, but at the same time might produce collisions between characters. Then, iteratively, we optimize the trajectories to handle collisions. We try to keep the produced trajectories as close as possible to the initial guessed trajectories, to minimize the magnitude of collision-avoidance maneuvers. How do we define this? Do we measure it? I don't think we do either.

To conclude, our main contribution is an optimization-based algorithm to compute high quality animation trajectories (2D global navigation trajectories) for individual crowd patches under constraints (expressed as a set of spatio-temporal boundary control points).

The remainder of this paper is organized as follows: Section 2 proposes a short overview on the state of the art. Section 3 details our technique to compute these internal trajectories. Then, in Section 4 some results, together with their performance and quality analysis are shown before a brief discussion and concluding remarks.

2 State of the Art

Most often, virtual environments are populated based on crowd simulation approaches [Thalmann and Raupp Muse 2013]. An ambient crowd is generated from a large set of moving characters, mainly walking characters. Recent efforts in crowd simulation have enabled dealing with great performances [Pettré et al. 2006; Treuille et al. 2006], high densities [Narain et al. 2009] or controllable crowds [Guy et al. 2009]. There has been a lot of effort to develop velocity-based approaches [Paris et al. 2007; van den Berg et al. 2007] which display much more smooth and realistic locomotion trajectories, especially thanks to anticipatory adaptation to avoid collisions between characters. Nevertheless, ...

Simulation-based techniques are ideal to create an ambient crowd for large environments. Several problems are recurrent with such approaches: a) crowd simulation is computation demanding, crowd size is severly limited for interactive applications on light computers; b) simulation is based on simplistic behaviours (e.g. walk, avoid collisions, etc.), it is difficult to show diverse and rich crowds based on classical crowd simulation algorithms; c) crowd simulation is prone to animation artifacts or deadlock situations, it is impossible to guarantee animation quality.

Example-based approaches attempt to solve the limitations on animation quality. The key idea of this approaches is that agents behave accordingly to set of existing example trajectories [Lerner et al. 2007; Ju et al. 2010; Charalambous and Chrysanthou 2014]. Locally, trajectories are necessarily of good quality, because they reproduce recorded ones. However, such approaches raise other difficulties: it is difficult to guarantee that the database of examples will cover all the required content, it can be difficult to control behaviors and interactions displayed by characters if the database content is not carefully selected. Finally, those approaches are also computationally demanding.

To solve both performances as well as quality issues, crowd patches were introduced in [Yersin et al. 2009]. The key idea is to generate an ambient moving crowd from a set of interconnected patches. Each patch is a kind of 3D animated texture element, which records the trajectories of several moving characters. Trajectories are periodic in time, so that the crowd motion can be played endlessly. Trajectories boundary conditions, at the geometrical limits of patches are controlled to be able to connect together to patches, with characters moving from one patch to another. Thus, a crowd animated from a set of patches have a seamless motion, patches limit cannot be easily detected. The boundary conditions are all registered into *patterns*, which are sort of gates for patches with a set of spacetime input/output points. Please refer to the Yersin's paper for details.

Nevertheless, using the crowd patches approach, it is important to

*e-mail:emails@emailplace

work with a limited set of patters to enable easily connecting various patches. As a result, it is important to be able to compose a patch by starting from a set of patterns, and to deduce internal trajectories of patches from the set of bounday conditions set by patterns. As a result, we need to compute trajectories for characters that pass through a given set of spatiotemporal waypoints (not that we consider 2D trajectories for characters global motion here). This problem is difficult. Indeed, generally, steering techniques for characters consider spatial goals, but does not consider the time a character should reach its waypoint. Dedicated techniques are required.

Yersin suggests using an adapted Helbing technique to compute internal trajectories [Helbing et al. 2005]. The key idea is to connect input/output points together with linear trajectories, to model characters as particles attracted by a goal moving along one of these linear trajectories, combined with repulsion forces to avoid collision. One problem with this approach is limited density level, as well as the level of quality of trajectories: they suffer the usual drawbacks of Helbing's generated trajectories, i.e., lack of anticipation, which result into non natural local avoidance maneuvers.

Compared to previous techniques to compute internal trajectories, we suggest formulating the problem of computing internal trajectories as an optimization problem. First, we suggest optimizing the way input and output points are connected. Especially, as waypoints are defined in space and time, we connect them trying to get some *good* walking speed (close to the average human walking speed). Indeed, characters moving too slow or too fast are visually evident artifacts. Second, after having connected waypoints with linear trajectories, we deform them to remove possible collisions. We do this through an iterative process trying to remove collisions with as limited as possible changes to the initial trajectories. We show improvements in the quality of results as compared to the original work by Yersin et al.

Would be nice to add reference and comparison with the rule based techinique. Completely agree with that ... I did not copied all the coments remaining in this Section. See google docs.

3 Method overview

Overview: remind definitions on crowd patches: patch, pattern, spatiotemporal waypoints (boundary conditions: input, output, initial states // boundary conditions should be strictly enforced. Movable control points), period of time . . .

Starting Definitions:

Patch - A patch is a set $\{A, \pi, D, S\}$ where A is a geometrical area with a convex polygonal shape, π the period of time of the animation, D and S are the sets of dynamic and static objects, respectively. These last two sets may be empty in case of an empty patch.

Static obstacles - Static objects are simple obstacles whose geometry is fully contained inside the patch.

Dynamic objects Dynamic objects are animated: they are moving in time according to a set of trajectories T.

We define a **trajectory** inside a patch as a function going from time to position, more specifically from the subset $[0, \pi]$ to A:$\tau : [t_1, t_2] \rightarrow A, \quad 0 \leq t_1 < t_2 \leq \pi$. We represent a trajectory as a list of control points connected by segments.

A **control point** is a point in space and time. All control points in a trajectory will either be a movable control point or a boundary control point. Boundary control points serve as entry and exit points to the patch and cannot be moved, added or deleted. Movable control points can be moved, added, or removed from the trajectory as long as they do not violate the constraints of the patch, their position must be inside A and the time between t_1 and t_2.
– Should we state exactly whate these constraints are here?
– Sure, I added a line for that

A **segment** is a straight line connecting two control points in a specific order. Since these are unidirectional lines in space-time, it is important to remember that they are not allowed to go backwards in time.

There are two categories of dynamic objects: endogenous agents and exogenous agents. **Endogenous agents** remain inside A for the total period of time π. In order to achieve periodicity for the animation, they are associated with a trajectory $\tau : [0, \pi] \leftarrow A$, such that it respects the periodicity condition: the position at the start and at the end of the animation must be the same, i.e. $\tau(0) = \tau(\pi)$. **Exogenous agents** go outside A. They enter the patch at time $t_{initial}$ and position $a_{initial}$, and they exit at time t_{final} and position a_{final}. For each agent we associate a sequence of n trajectories $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Sequences may have only one trajectory, but some agents require additional trajectories in order to satisfy speed and time constraints. The following conditions must be respected in each sequence of trajectories associated with an exogenous agent:

- $a_{initial}$ and a_{final} must be points in the border of A. Otherwise, they couldn't be exogenous agents.
- If the sequence is composed by more than one trajectory, for each two contiguous trajectories, the following must be true to ensure continuity: $\tau_i(\pi) = \tau_{i_{next}}(0)$.

Note that the second condition implicitly implies that in sequences with multiple trajectories, each middle trajectory must be fully defined in the period of time $[0, \pi]$, while τ_1 must be defined in $[t_{initial}, \pi]$ and τ_n must be defined in $[0, t_{final}]$.

Pattern - If a patch is a spatio-temporal cube (or any other right prism, depending on the type of polygon used as its area, then a pattern could be defined as one lateral side of the cube (or right prism). Specifically, it is a rectangle whose base is one of the edges of the polygonal area (we define I as this two dimensional vector), and its height is equal to the period. These patterns also include the sets of boundary control points. We divide this set evenly into two subsets: Input and Output. The Input set contains the boundary control points where exogenous agents begin their trajectories; we call these Entry Points. Conversely the elements of Output are called Exit Points. They establish the position in time and space that the exogenous agents finish their paths. Formally defined, a patch P is:

$$P = l, p_i, I_i[p_i, t_i], O_j[p_j, t_j]$$

We populate virtual environments by sticking patches together. Thus, we have to ensure continuity between trajectories for exogenous agents passing through two contiguous patches. This means that two adjacent patches must have a similar pattern on the side they share. The vector l Is this vector l or I?, please check and period must be equivalent and the sets of Input and Output are exchanged. If we have $P_1 = l, pi, I[p_1, t_1], O[p_2, t_2]$ and $P_2 = l, pi, I[p_2, t_2], O[p_1, t_1]$, then, in order to satisfy C^0 continuity we must ensure:

$$l = l, pi = pi, p_1 = p_1, t_1 = t_1, p_2 = p_2, t_2 = t_2.$$

We then say P_1 is the mirror pattern of P_2. In the animation, this will be seen as an agent going from one patch to an adjacent one. If the area of a patch is a square, the patch defines 4 patterns, one for each of its sides. Patterns defined by a patch have the property that the sum of the cardinality of all the Inputs is the same as the sum of the cardinality of all Outputs. We call this the parity condition: $\sum(|Inputs|) = \sum(|Outputs|)$.

A patch defines a set of Patterns, and conversely, a set of patterns satisfying the parity condition, having the same period, and whose vectors define a convez polygonal area, can be used to create a patch.

4 Results

5 Discussion

6 Conclusions

References

CHARALAMBOUS, P., AND CHRYSANTHOU, Y. 2014. The PAG crowd: A graph-based approach for efficient data-driven crowd simulation. *Computer Graphics Forum*.

GUY, S. J., CHHUGANI, J., KIM, C., SATISH, N., LIN, M., MANOCHA, D., AND DUBEY, P. 2009. Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, ACM, SCA '09, 177–187.

HELBING, D., BUZNA, L., JOHANSSON, A., AND WERNER, T. 2005. Self-organized pedestrian crowd dynamics: Experiments, simulations, and design solutions. *Transportation Science 39*, 1 (February), 1–24.

JU, E., CHOI, M. G., PARK, M., LEE, J., LEE, K. H., AND TAKAHASI, S. 2010. Morphable crowds. In *Proc. of ACM SIGGRAPH Asia*, ACM, SIGGRAPH Asia '10, 140:1–140:10.

LERNER, A., CHRYSANTHOU, Y., AND LISCHINSKI, D. 2007. Crowds by example. *Computer Graphics Forum 26*, 3 (September), 655–664.

NARAIN, R., GOLAS, A., CURTIS, S., AND LIN, M. C. 2009. Aggregate dynamics for dense crowd simulation. In *Proc. of ACM SIGGRAPH Asia*, ACM, SIGGRAPH Asia '09, 122:1 – 122:8.

PARIS, S., PETTRÉ, J., AND DONIKIAN, S. 2007. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum 26*, 3 (September), 665–674.

PETTRÉ, J., CIECHOMSKI, P., MAM, J., YERSIN, B., LAUMOND, J.-P., AND THALMANN, D. 2006. Real-time navigating crowds: Scalable simulation and rendering. *Computer Animation adn Virtual Worlds 17*, 3–4, 445–455.

ROCKSTAR-GAMES, 2004. Grand theft auto: San andreas. http://www.rockstargames.com/grandtheftauto/, October.

THALMANN, D., AND RAUPP MUSE, S. 2013. *Crowd Simulation*, 2nd ed. Springer.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *Proc. of ACM SIGGRAPH 2006*, ACM, SIGGRAPH '06, 1160–1168.

VAN DEN BERG, J., LIN, M., AND MANOCHA, D. 2007. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, IEEE, ICRA '07, 1928–1935.

YERSIN, B., MAÏM, J., PETTRÉ, J., AND THALMANN, D. 2009. Crowd patches: Populating large-scale virtual environments for real-time applications. In *Proc. of the 2009 Symp. on Interactive 3D Graphics and Games*, ACM, I3D '09, 207–214.

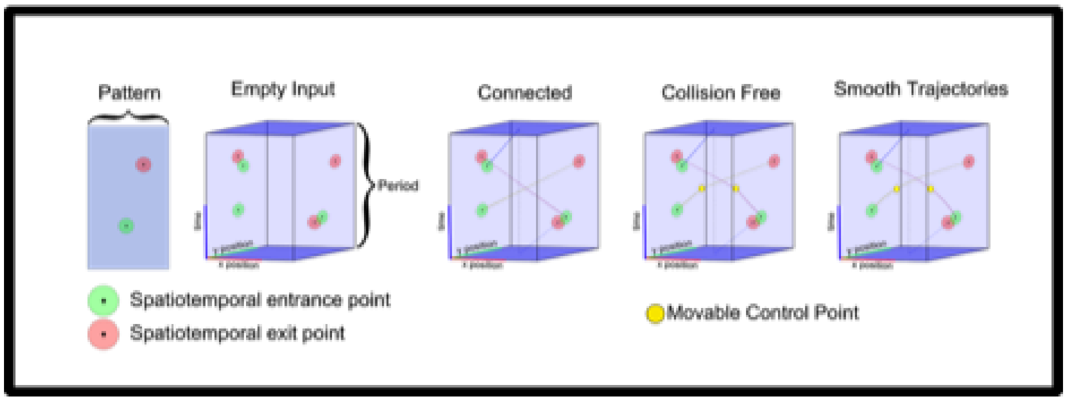

Figure 1: *Caption caption caption caption caption caption caption caption caption caption*