Interactive Crowd Simulation on Mobile Devices in an Augmented Reality Environment

Aytek Aman
Bilkent University
aytek.aman@cs.bilkent.edu.tr

Ateş Akaydın
Bilkent University
akaydin@cs.bilkent.edu.tr

Uğur Güdükbay
Bilkent University
gudukbay@cs.bilkent.edu.tr

Abstract

We propose a technique to do interactive crowd simulation on mobile devices within an Augmented Reality (AR) framework. We localize the camera in a prepared environment that includes a set of natural markers. We use least squares optimization to obtain extrinsic camera parameters. During tracking, the users can create paths for the agents to follow through touch inputs. User-specified paths are converted to Bézier curves. We use navigation mesh-based planning and the user-specified paths interchangeably for global path planning.

Keywords: augmented reality, virtual reality, crowd simulation, virtual worlds, path planning

1 Introduction

Simulation of crowds in real-time has been a major field of study for researchers. So far, crowd simulation is mostly performed in virtual environments. With the increased capabilities of the modern computing hardware, it is now possible to simulate crowds in AR environments where virtual objects are seamlessly combined with the real world.

This study proposes an AR framework where it is possible to simulate crowds in real world with the help of the mobile devices. To the best of our knowledge, there are no studies in the literature using AR for interactive crowd simulation on mobile devices. The proposed system is original in this respect.

2 Related Work

In recent years, vision-based tracking methods became increasingly popular. Vision-based tracking methods generally depend on feature extraction / tracking. Both edges and points can be used for tracking. Drummond et al. [1] use model edges for tracking. The work of Wuest et al. [2] is another good example of vision-based methods.

Crowd simulation methods may differ with respect to the density and size of the virtual crowd to be simulated. Pelechano et al. [3] made a comprehensive survey on virtual crowd simulation techniques.

3 Proposed Technique

3.1 Camera Localization

To locate the camera, we use a model-based tracker. We first build the coarse model of the simulation space. We then preprocess this model to extract the salient edges. The salient edges of the model must be recognizable in the video feed. We mark an edge as salient if the

angle between the normal vectors of the neighboring triangles is above a certain threshold. The tracking algorithm is based on the work of

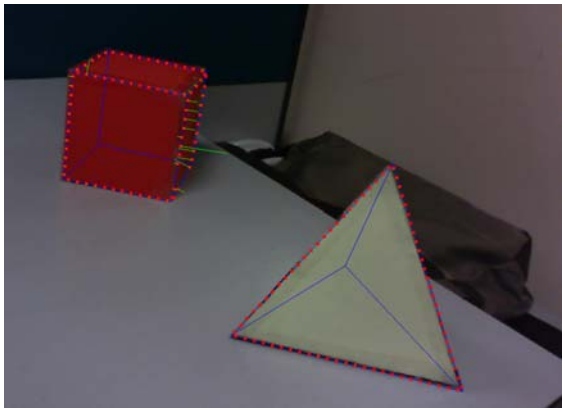

Figure 1: Camera localization during tracking.

Drummond et al. [1]. When a video feed is updated, the camera parameters are calculated and assigned to the virtual camera. Firstly, model edges are clipped to the view frustum. Clipped edges are then projected onto the camera image. Edges are sampled at regular intervals. Then, the visibility of the sample points are determined using ray casting. For each visible sample point (shown red in Figure 1), a discontinuity is searched along the direction of the edge normal. Once such a point is found, its distance (shown green in Figure 1) to the corresponding sample point is calculated. The tracker finds a transformation matrix that minimizes total distances between the sample points and their counterparts in the image using a least squares solver.

To increase the accuracy, we use RANSAC [4] to eliminate outliers. Outlier elimination is performed per edge assuming that sample points should be matched with points that lie on the same line in the image. Outliers are excluded from the correspondence calculations.

3.2 Interactive Path Planning

While the camera is localized, the user can draw paths using touch based interaction. Successive interaction points obtained with ray-casting start and continue paths by adding new nodes (Figure 2). We fit a piecewise cubic Bézier curve by using the user-specified control points ($\vec{p}_i$). Sub-nodes ($\vec{s}_j$) are then sampled from

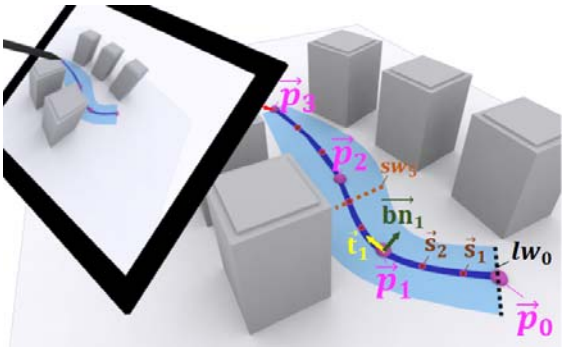

Figure 2: Path creation and manipulation.

the curve equation in between pairs of control nodes. We define curved lane on the path with varying lane widths (lw_i) at control node positions. We linearly interpolate sub-node lane widths (sw_j) with respect to the two neighboring control nodes. The user can modify the curved lane by changing control node positions ($\vec{p}_i$), binormal vectors ($\vec{bn}_i$) and lane width (lw_i) associated with the control nodes.

3.3 Crowd Simulation

Virtual agents are instantiated at positions, which we call *agent sources*. Similar to agent sources, we also introduce agent sinks to the system. Sinks represent the global goals of the agents and agents achieving their goals are automatically removed from the system. For local planning, we use the Reciprocal Velocity Obstacles (RVO) method proposed by Berg et al. [5]. For global path planning, we make a decision between the path provided by the NavMesh [6] and the path(s) specified by the user. When the user finishes positioning of the agent sinks and sources, we initialize a routing table ($R_{src,snk}$) to select the best path (P) among the set of user-specified paths (S).

$$R_{src,snk} = \underset{P \in S}{\operatorname{argmin}}\, \delta(P, \vec{n}_{src}, \vec{n}_{snk}) \quad (1)$$

In Equation 1, $\delta(P, \vec{n}_{src}, \vec{n}_{snk})$ represents the estimated path length from the agent source position ($\vec{n}_{src}$) towards the agent sink position ($\vec{n}_{snk}$) while traveling on path P. This distance

is estimated using Equation 2.

$$\begin{aligned}\delta(P,\vec{n}_{src},\vec{n}_{snk}) &= d(\vec{n}_{src},\vec{s}_{c_{src}}) \\ &+\alpha\beta(P,c_{src},c_{snk}) \\ &+d(\vec{s}_{c_{snk}},\vec{n}_{snk}) \\ c_{src} &= f(P,\vec{n}_{src}) \\ c_{snk} &= f(P,\vec{n}_{snk})\end{aligned} \quad (2)$$

In Equation 2, $\vec{n}_{src}$ and $\vec{n}_{snk}$ are the positions of an agent source and sink, $d(\vec{x},\vec{y})$ is the Euclidean distance between the two arbitrary coordinates, $\beta(P,c_0,c_1)$ is the distance between the sub-nodes identified by indices c_0 and c_1 on path P, α is the weight which decides the preferability of the user defined paths and the NavMesh, c_{src} and c_{snk} are the indices of the sub-nodes that are closest to $\vec{n}_{src}$ and $\vec{n}_{snk}$ on path P, respectively. Function $f(P,\vec{x})$ retrieves the index of the sub-node on P that has the shortest distance to the given arbitrary point $\vec{x}$ (Equation 3).

$$f(P,\vec{x}) = \{c|\forall k : d(\vec{s}_c,\vec{x}) \leq d(\vec{s}_k,\vec{x})\} \quad (3)$$

In Equation 4, the β function estimates the length of the curve segment between sub-node indices c_{src} and c_{snk}.

$$\beta(P,c_{src},c_{snk}) = \sum_{i=c_{src}}^{i\neq c_{snk}} d(\vec{s}_i,\vec{s}_{i+1}) \quad (4)$$

During instantiation, each agent is assigned a random agent sink as the global target. Instantiated agents query the routing table ($R_{src,snk}$) once to find the shortest user-specified path from their instantiator source to the targeted sink. The agent then compares the path length ($\delta(P,\vec{n}_{src},\vec{n}_{snk})$) of the user-specified path with the length of the path generated by the underlying NavMesh. The shorter path becomes the agent's path to follow. While agents are following a single path, they are queued into a single curve, which is an unnatural behavior. To remedy this problem we distribute the agents over lanes specified around the paths. Assuming an agent is between the sub-node positions $\vec{s}_i$ and $\vec{s}_{i+1}$, we compute the local target of the agent as described in Equation 5.

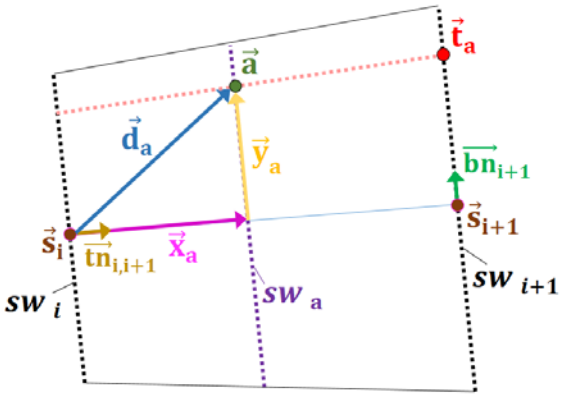

Figure 3: An agent following a lane.

$$\begin{aligned}\vec{tn}_{i,i+1} &= \frac{\vec{s}_{i+1}-\vec{s}_i}{\|\vec{s}_{i+1}-\vec{s}_i\|} \\ \vec{d}_a &= \vec{a}-\vec{s}_i \\ \vec{x}_a &= (\vec{d}_a \bullet \vec{tn}_{i,i+1})\vec{tn}_{i,i+1} \\ \vec{y}_a &= \vec{d}_a - \vec{x}_a \\ sw_a &= (\frac{\|\vec{x}_a\|}{\|\vec{s}_{i+1}-\vec{s}_i\|})sw_{i+1} \\ &+(1-\frac{\|\vec{x}_a\|}{\|\vec{s}_{i+1}-\vec{s}_i\|})sw_i \\ r &= min(2\frac{\|\vec{y}_a\|}{sw_a},1)\end{aligned} \quad (5)$$

In Equation 5, $tn_{i,i+1}$ is the segment's tangent vector, $\vec{d}_a$ is agent position with respect to the first sub-node, sw_a is the interpolated lane width at the agent's position a. The local target $\vec{t}_a$ is computed using Equation 6 as seen in Figure 3.

$$\vec{t}_a = \vec{s}_{i+1} + \frac{\vec{d}_a \bullet \vec{bn}_{i+1}}{|\vec{d}_a \bullet \vec{bn}_{i+1}|}(\frac{rsw_{i+1}}{2}\vec{bn}_{i+1}) \quad (6)$$

When the agent achieves $\vec{t}_a$, it proceeds to the next segment. When the agent reaches the closest sub-node to the targeted sink node, it leaves its current path and starts following the NavMesh.

4 Results

We evaluated our approach on two different hardware configurations. The first configuration includes a workstation with Intel® i7 processor (2.80 Ghz), 16 GBs of RAM, and an NVIDIA® Quadro K3000M (2 GB) GPU. In the second configuration, we use a Samsung Galaxy Note 10.1 (P600) model android based tablet device.

Table 1: The average time spent on tracking and rendering tasks and the average frame rates for different number of agents. Tracking time is computed using five markers with twelve salient edges each. Tracking and rendering times are in milliseconds.

Config.	Tracking	Rendering					
		20 Agents		40 Agents		60 Agents	
		comp. time	frame rate	comp. time	frame rate	comp. time	frame rate
PC	3.2	0.2	600	0.32	360	0.40	260
Mobile	10.5	2.0	37	3.44	25	5.2	18

The average computation times spent on tracking and rendering; and frame rates are given in Table 1. Tracking algorithm is ran only when the video feed is updated (at 25 Hz) for a standard web-camera. Majority of the cost arises from rendering alone. A snapshot of the application is given in Figure 4.

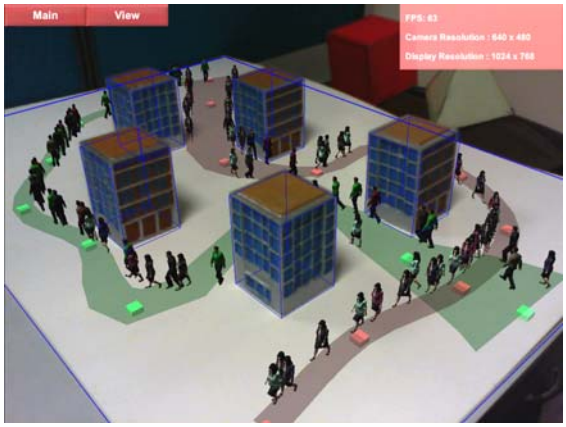

Figure 4: A snapshot of the working application.

5 Conclusion

We proposed an approach to do virtual crowd simulation on mobile devices in an AR environment. To the best of our knowledge, our work is the first real-time approach which robustly integrates crowds into AR environments. We believe that our work is particularly useful for applications of architectural visualization. For example prototype construction models are usually built as table-top miniature models at first. Our approach enables creating of different visual experiences in such miniaturized models.

6 Acknowledgements

This work is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant No. 112E110.

References

[1] Tom Drummond, Ieee Computer Society, and Roberto Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:932–946, 2002.

[2] Harald Wuest, Florent Vial, and Didier Stricker. Adaptive line tracking with multiple hypotheses for augmented reality. In *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '05, pages 62–69, Washington, DC, USA, 2005. IEEE Computer Society.

[3] N. Pelechano, J.M. Allbeck, and N.I. Badler. *Virtual Crowds: Methods, Simulation, Control and Synthesis*, volume 8. Lectures on Computer Graphics and Animation, Morgan and Claypool Publishers, 2008.

[4] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[5] Jur P. Van Den Berg, Dinesh Manocha, and Ming C. Lin. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *International Conference on Robotics and Automation*, pages 1928–1935, 2008.

[6] Greg Snook. Simplified 3D Movement and Pathfinding Using Navigation Meshes. In *Game Programming Gems*, pages 288–304. Charles River Media, 2000.