Crowd Simulation Rendering for Web

Daniel P. Savoy, Marcio C. Cabral, Marcelo K. Zuffo
Interdisciplinary Center in Interactive Technologies (CITI-USP), University of São Paulo*

Figure 1: *Real-time (60fps) rendering of a pre-computed crowd simulation. All 8,000 agents are using skinning and skeleton animation on GPU, with no levels of detail or impostors.*

Abstract

Simulation and rendering of large crowds are very demanding tasks on computational resources and until recently were inconceivable to be performed by a web browser. However, with the increasing capacity of GPUs and the maturation of web front-end development, could a web-based simulation of massive crowds be achieved in real-time in today's web-browsers? In this work we present the implementation of a minimal visualization tool for crowd simulation results, with the ability of rendering thousands of animated agents in real-time using WebGL. We also briefly present some current challenges of accomplishing crowd simulations in a web environment.

CR Categories: I.3.6 [Computing Methodologies]: Computer Graphics - Methodology and Techniques.

Keywords: Crowd Simulation, Crowd Rendering, Instancing.

1 Introduction

Crowd simulation focus on the collective behavior of pedestrians, and very often it requires simulating the individual behavior of a large number of independent virtual agents and their interactions. The computational needs for this kind of application grow quickly with the complexity of each agent and with the total number of simulated agents. The same can be said about crowd rendering, which also demands a lot of computational resources, especially from the GPU. Therefore, simulation and visualization of large crowds has been restricted to professional users with high performance equipment and specialized software applications.

*email: {dsavoy, mcabral, mkzuffo}@lsi.usp.br

Web3D '15, June 18 - 21, 2015, HERAKLION, Greece
ACM 978-1-4503-3647-5/15/06.
http://dx.doi.org/10.1145/2775292.2778302

Taking into account the modern browsers capacity to access the graphics card, run parallel processes, and use binary data transfer, what we hope to demonstrate is that the current web environment is mature enough to support a new generation of crowd simulation and visualization applications.

2 Related Work

There has been constant research activity during the last years on crowd simulation and rendering, as new level of detail methods, impostors, virtual reality, collision algorithms, behavior modeling, and others techniques [Thalmann and Musse 2013; Ali et al 2013] that surely could be used on web-based crowds. But, regardless of the fact that the possibilities of web-based simulations are subject of interest for more than a decade [Kuljis and Paul 2001], just a few works have dealt with crowds in the limited web-browser environment [Bruzzone and Signorile 1999; Klein et al. 2014].

3 Implementation

In the present work, our main goal was to create a real-time crowd rendering of thousands of agents. As an initial step we developed a basic rendering engine while studying some potential approaches to reach real-time performance and a large number of agents to depict dense crowd scenarios. In parallel, we explored a few alternatives on how to obtain simulation data at the right speed rate to feed the real-time rendering engine.

3.1 Naive Approach

Initially the simulation itself was entirely calculated on the client, in real-time. To do so, we ported to JavaScript the RVO2 collision avoidance library to run on a parallel process. The RVO2 library has been developed by the University of North Carolina and is based on their ORCA method [van den Berg et al. 2010]. Our ported version worked very well for small simulation scenarios, like an airplane boarding, where the crowd had just a few hundred agents, but larger crowds could not be simulated in real-time. For the rendering we made use of ThreeJS, but the lack of hardware instancing support in the library was a prohibitive factor to increase the number of agents, or even to use more elaborate 3D

models. Therefore, we decided to approach the problem using just native WebGL.

3.2 GPU Instancing

The ordinary approach to draw multiple 3D objects with WebGL is to use JS libraries to import 3D models, manipulate the parameters of each model, and then feed the individual models to the shaders, one by one. But, when we try to draw thousands of models at real-time, it simply is not going to be fast enough, because passing data from CPU to GPU is a costly operation and Javascript has an inferior performance compared to native C/C++ and, on top of that, browsers offer no thread support. But the shaders on the graphics card can be orders of magnitude faster than JavaScript code, so a GPU instancing mechanism is a *sine qua non* condition to perform rendering of large crowds in a browser. With instancing, we can feed a model and parameters only once to the GPU and draw it multiple times. Luckily, hardware instancing started very recently to be supported on most of web browsers through the ANGLE instanced arrays extension [Kronos Group 2014].

3.3 Levels of Detail and Multiple Processes

Using GPU instances we were finally able to render up to ten thousand 3D humanoid models. Alternatively, we could render several tens of thousands of simpler models like the ones we used in our earlier simulations. To get the better of those two options, we decided to implement a level of detail (LOD) engine, to sort the position of agents and use better models to display the ones near to the camera. To achieve that, we created a pool of processes to calculate the position of everyone, perform frustum clipping, and split the agents in different levels of detail.

At this point, we were using a simple random walker as the behavior model of the agents (just to make them move at real-time), and the camera position was also in constant movement, linked to mouse events, implying that the LOD needed to be resorted all the time, at 60fps. All this necessary communication passing between processes became an issue, mainly because the Webworker API does not provide a shared memory, and cloning large arrays of data was too slow. But, we found that some modern browsers had started to offer an interface called Transferable Objects that allows the ownership of some typed array objects to be transferred between process at almost no time, and this became our default message passing paradigm for all agent related data.

3.4 Animation and Skinning

Once the rendering was not a bottleneck anymore, we proceed to improve our 3D humanoid model with a skeleton, implementing skinning and skeletal animation in the shaders. It was a relatively easy step, but it gave a whole new level of realism to the crowd rendering. All that is necessary is generating a texture of floats to be feed to the GPU, containing encoded rotation quaternions of each bone at each key frame. Interpolation is achieved inside the shaders, with agent speed and state as parameters. We were able to render around 9,000 of such animated agents at 60 fps using a 25-bone skeleton (no LOD) using a NVidia GTX 660.

3.5 Incomplete Web-Based Crowd Simulation

Despite of the success implementing the crowd rendering, the crowd simulation of thousands of agents was too computationally intensive for our Javascript port of RVO2 to be able to handle in real-time. It was running in a separated process since our first few experiments, but the separated process itself lacked of local parallel processing capabilities, because in Chromium based browsers nested WebWorkers are currently not allowed.

As an alternative, we used our simulation process as the base for a complementary and separated offline simulator, that saved all generated agent position data in JSON text files, for later utilization. It took several hours to compute a 7-minute simulation of 5,000 agents with 60 steps per second. The resulting text files were humongous, occupying nearly 4Gb on disk. So, even if we ignore the time needed to compute the simulation, the volume of data would not change, what is unacceptable in terms of network traffic. In the end, we were obligated to develop a file reading and data caching parallel module to feed the rendering engine at a constant rate, and also protect the browser from reaching its memory capacity limit.

4 Conclusion and Future Work

In this work we presented the implementation of a visualization tool that is able to successfully render, in real-time, large crowds of thousands of complex animated 3D agents, running on an ordinary computer, and using only standard supported features of modern browsers. Our results strongly suggest that the path towards web-based interactive crowd simulation is already open, and this kind of application can become a reality in a short period of time from now. There still is a lot of work to be done to improve the rendering engine (animation, realism, etc), but the most pressing points are the simulation data volume and processing time. Both still open issues that one must overcome in order to have interactive large crowd simulations in a web environment in real-time.

Acknowledgements

This work has been partially supported by ME/CNPq (process number 487528/2013-1).

References

ALI, S., NISHINO, K., MANOCHA, D., & SHAH, M. 2013. Modeling, Simulation and Visual Analysis of Crowds. Springer New York. doi:10.1007/978-1-4614-8483-7

BERG, J. VAN DEN, GUY, S., LIN, M., & MANOCHA, D. 2011. Reciprocal n-body collision avoidance. In *Robotics Research*. doi:10.1007/978-3-642-19457-3_1

BRUZZONE, A G., & SIGNORILE, R. 1999. Crowd control simulation in Java based environment. *Simulation Series*.

KLEIN, F., SPIELDENNER, T., SONS, K., & SLUSALLEK, P. 2014. Configurable instances of 3D models for declarative 3D in the web. In *Proc. of WEB3D 2014*. doi:10.1145/2628588.2628594

KRONOS GROUP. WebGL ANGLE_instanced_arrays Khronos Ratified Extension Specification. 2014.

KULJIS, J., & PAUL, R. J. 2001. An appraisal of web-based simulation: Whither we wander? *Simulation Practice and Theory*. doi:10.1016/S0928-4869(01)00032-5

THALMANN, D., & MUSSE, S. R. 2013. Crowd Simulation (2nd Ed). Springer London. doi:10.1007/978-1-4471-4450-2