Real-Time Rendering of Densely Populated Urban Environments

Franco Tecchia
Laboratorio Percro - Scuola Superiore S. Anna, Pisa, Italy

Yiorgos Chrysanthou
Department of Computer Science, University College London, London, UK

Abstract. In this paper we present some preliminary results concerning a real-time visualisation system for densely populated urban environments. In order to be able to render the large number of humans without compromising too much the image quality, we developed a method based on Image-Based Rendering techniques. To allow them to move freely in the city while avoiding collisions against the environment and other humans, we developed a simplified collision test that makes use of the graphics hardware to quickly generate a discretization of the environment. Although our research is at an early stage, the results are already quite promising; we are able to render in real-time a virtual city with thousands of walking humans on a standard PC. Several avenues for further investigation are finally proposed.

1 Introduction

Recently we have seen the appearance of many geometric models of towns and cities around the world with an enormous number of potential applications. There has been a lot of work in accelerating the rendering of such models. Many different techniques have been used such as visibility culling [3] and image based rendering [6, 11], to the point where we can now render quite substantial models in real-time. At UCL we have a polygonal model of London that covers a large portion of the city, 160 km in total, with some parts of it modelled in great detail. This model was original developed through the COVEN project [4] for use in various VR simulations. However, without any humans populating it, it did not look very real. Our overall objective is to render in real-time the virtual model of London, simulating also the crowds and traffic of the city. There are several problems to be addressed before this can be realised, such as rendering, simulating the behaviours, avoiding collisions both among the crowd and with the environment. In this paper we will concentrate mainly on fast crowd rendering and collision avoidance, the implemented behaviour is random, constrained only by collisions.

The algorithms we will describe are better viewed in the context of a level-of-detail system where different techniques are applied in order to produce the best possible image while maintaining a high frame rate. At the lowest level, when the viewer is some distance above the crowded city and potentially has in view a very large number of objects and moving humans, the techniques described here can be used. As the viewer gets closer some individuals become more visually important but at the same time the number of visible objects becomes smaller. Higher detail solutions should be used as the avatars get too close [14, 12].

In Section 2 we describe the rendering of the avatars and in Section 3 we touch briefly on the collision solution. Some implementation details and results are presented in Section 4 followed by conclusions and some proposals for future directions and improvements.

2 Rendering the virtual humans using image based rendering

The main difficulty in the rendering process of the crowd resides in the high number of independent elements to visualise and animate. Clearly using detailed geometric representations will not scale enough. An attractive alternative is the use of image based rendering (IBR) techniques [9, 10]. Aubel et al. [1] applied such ideas recently to virtual humans. However, they use dynamically computed impostors which can only be used for a few frames before being discarded.

The basic idea of our approach is to make use of only one polygon per human, using a pre-computed texture in a way that it shows a good approximation to the human aspect as seen from a given viewpoint. Taking in account that the amount of dedicated texture memory on low-cost workstations is increasing at an exponential rate in the recent years we decided to make wide use of it in order to store a high number of different views of a human character sampled while performing a walking animation. We decided not to use interpolation between views as that would be too cpu-intensive for our purpose. Instead, we opted for a simple solution that allows us to take the maximum advantage of the graphics hardware available today on the market.

The first phase of the algorithm consists of the generation of a sufficiently large set of sample views of an object. An example is shown in Figure 2.

In our tests we produced 16*8 different images. Each row of images represents the object as seen from 16 different viewpoints at a certain height, rotating around the Y axis. Mirroring them we are able to generate 32 different views of the object with a difference of 11.25 degree one from the other. Each row of images corresponds to a different elevation of the viewpoint. The images, inclusive of the appropriate alpha values, are stored as textures, and the visualisation of every single human is obtained by pasting one of these images to a single polygon having the right orientation toward the camera.

This kind of approach might at first appear to be very expensive in terms of memory requirement. Fortunately, when applied to the visualisation of a virtual crowd there are some constraints that allow us to reduce the number of necessary images:

1. In scenes that show a high number of humans, the vast majority of them appear to be far from the viewpoint (see Figures 5 and 6.)
2. Every human can have his own movement direction but the possible rotations of each human is usually limited to the y-axis.
3. We will rarely have a view of some human as seen from below, so we can undersample these views.

At each frame, to select the correct image from the set of the different images available we disctretize the direction between each human and the viewpoint (Figure 1). From this process we get two indices (one referring to the elevation of the viewport over the human, and the other referring to the direction around the human) that permit us to select the right row and column of the image stored. Adding an integer number to the column index when retrieving the right image permit us to simulate a rotation of a human around the y-axis with intervals of 11.25 degrees. As the human-camera direction changes we select different images from the set that we have, so as to get the better possible approximation.

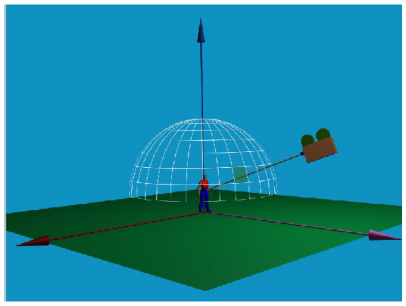


Fig. 1. Discretising the view direction between the object and the viewpoint

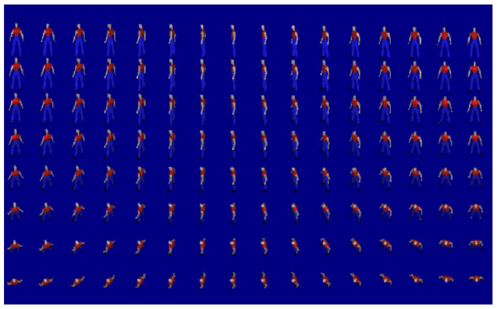


Fig. 2. Views of the human from the set of discrete directions

Storing in the texture memory snapshots from different frames of an animation for the same model and selecting different textures at each frame we are then able to visualise animated figures. The result of the animation can be seen in Figure 6.

Our algorithm result to be fast enough to allow the visualisation of virtual environments populated by thousands of humans in real time. Because of the limited number of memorised images and the absence of procedures to warp them the visualisation turns out to be slightly imprecise and if one focuses his attention on a single human it is sometimes possible to perceive a *popping* effect in the passage between two different images. However, when the scene is populated, like in our case, with thousands of animated individuals, or if the viewpoint is sufficiently far, the popping effect becomes unnoticeable.

3 Collision avoidance using rasterisation

We need to consider two factors for the collision method. The first is the huge number of moving entities and second is the large (and rapidly increasing) complexity of the static model. Thus the algorithm needs to be fast and scalable; ideally the speed of the collision test should not depend on the polygonal complexity of the model. Space discretization methods [7, 8] are probably the most likely to satisfy these criteria. Graphics rasterisation hardware can be used to make the discretisation easy to implement and fast.

With an outdoor model such as a city these algorithms are well suited since the environment can be seen as a 2D plan with heights[1]. In [13] we described an approach that takes advantage of that property. Briefly, a *height-map* is created by taking an orthographic rendering of the static model with the camera looking down from above and then reading the z-buffer. This map constitutes a discreet representation of the height at each point in the environment and it is maintained in main memory as an array. At each frame we can test for collision by just mapping the position of each avatar on to this array and comparing the elevation of the avatar against the corresponding value on the height-map. If the difference between them is above a threshold, it means that the step necessary in order to climb up or down from the actual avatar position is too big and we classify the object as “unclimbable” forcing the avatar to change direction. Otherwise we allow the avatar to move to the new position and update its height using the value stored in the height-map.

[1]There are exceptions such as bridges and under-passes but these can be dealt with by adding another 2D level of rasterisation

Here we extend the idea of discretisation to inter-avatar collision, using another map which we call *collision-map*. However this is an integer array and it is not necessarily of the same resolution as the height-map. It is used not only for avoiding direct collision but also for keeping the avatars from coming too close to each other. The allowed proximity is determined by the size of the map pixels. Initially the values of all pixels in the map are set to zero. A pixel with a value zero indicates a free position in space while one with value > 0 is considered occupied (or too close to someone else). For each avatar its position in the map as well as its neighbourhood (the eight positions surrounding it) are found and their value incremented by one. When an avatar is to move, the target position is sampled. If it is already occupied then a different direction has to be chosen, otherwise it is allowed to move there. The move is done in two steps, first the current pixel position and neighbourhood are decremented by one and then the new position and neighbourhood are incremented by one. Notice that the neighbourhood of avatars are allowed to overlap (and thus a map pixel might have a value greater than one), however avatar-neighbourhood overlaps are not allowed. In this way avatars can come to within one map pixel away but no closer.

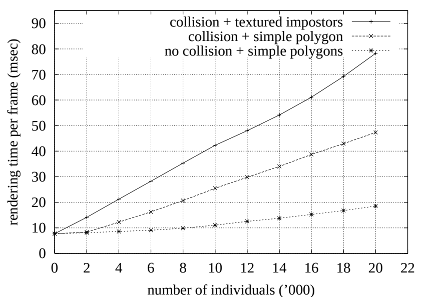
4 Implementation and Results

We developed the system using a low-end Intel workstation equipped with a 350 Mhz PentiumII processor, 256Mb RAM and an OpenGL compliant videocard with 32 MB video memory. We tried to maximise the rendering efficiency on this kind of architecture by following some basic principles. In particular we tried to limit the number of texture changes during the simulation, since this operation is very expensive in terms of time. To achieve this we grouped together all the different view images of an object for each animation frame into one single texture. In this particular implementation each image has a resolution of 64*64 pixels - we found this to be the best compromise in terms of image quality produced and memory requirements - therefore a texture of size 1024*512 pixels can hold all 16*8 views.

The animation stored was composed of 10 different frames of a walking human, produced using two commercial package: Kinetix 3DStudioMax and MetaCreations Poser. To build the single images we rendered the polygonal version of the humans from the sample directions and then copied every rendered image from the framebuffer into the texture memory, using alpha values for the transparent zones. Since we were not attempting to get the highest precision impostors and in order to reduce memory occupancy we store them with just 16 bits - 5 for each of R, G and B and 1 bit for A. In this way we need 1 Mb of uncompressed texture memory to store each single frame.

To test the rendering and the collision detection we run two tests, the results of which can be seen in Figures 3 and 4. In both graphs the Y axis shows milliseconds per frame and all three curves include the rendering of the static model and the impostor polygon for each avatar. However for the lower curve both collision and texturing of impostors is turned off, for the middle curve collision is turned on and for the higher one both collision and textured impostors are enabled.

For the first test we used a small city model of 2368 polygons. The aim was to measure the scalability of the method when the number of simulated individuals increases. With the second test we want to measure the scalability with regards to the polygonal complexity of the virtual environment. We kept the population steady at 10,000 avatars and varied the static model from a simple box like city block of 267 polygons to a fraction of the London model available here at UCL composed of 32128 polygons. As we can see from both tests we got almost linear curves which re-enforces our argument of



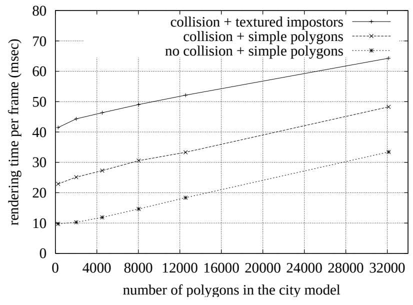


Fig. 3. Time to render against increasing number of particles

Fig. 4. Time to render against increasing complexity of the city model

scalability for both the avatar rendering and the collision. Just to read out one sample from Figure 3 above, a model with 2368 polygons using full collision detection and 10,000 textured impostors runs at 25 frames per second.

In terms of artifacts due to the discrete nature of the impostors, such as popping for example, they turned out to be scarcely notable because of the high number of animated figures visualised at the same time and the limited dimensions of these on the screen. This can also be verified from accompanying video.

5 Conclusions and future work

Using the algorithms described is possible to populate virtual environments with thousands of animated virtual humans in real-time even on low end machines. However there is great scope for extension and improvement.

In terms of rendering, a disadvantage of our algorithm resides in the big amount of texture memory necessary for storing the avatar views. One way to improve the method is rationalising the memorisation process of the set of views in a single texture. The current strategy relies on a regular subdivision of the texture; in this way there is a large portion of unused space in the final image (see Figure 2). An alternative is to use an irregular subdivision of the texture so as to be able to store more views in the same amount of space. We expect that using a smaller portion of the texture will produce also benefits on the fill-rate requirement for the visualisation of the crowd. It would be also very interesting to try our algorithm using some of the devices presents on the market that employ compression algorithms on the texture memory; on these devices is possible to reach compression ratios in the order of 6:1.

At the moment all the avatars have the same colour. One way of avoiding this is to use more than one textured polygon per avatar. For example one for each body member (hand, leg, torso, head), along the spirit of [1]. This will of course increase the rendering time however it has several potential benefits. The different parts could be interchanged between avatars, their color can be modulated during the frame providing more variety and they can also be packed more tight saving texture memory.

Adding behaviour would benefit the system immensely however it can also be quite expensive [14, 12]. At a low level of detail, we could at-least achieve more realistic walking patterns by making use of the pedestrian movement data made available, for example, by the UCL Bartlett School of Architecture [2]. These provide the density of the pedestrians along each pavement at each hour of the day. We can be represented the data as color on polygons corresponding to the pavements. These color polygons can then be rendered with an orthographic projection to provide a *density-map* of the scene. Avatars can then move and decide their paths by probabilistically sampling this map. The behaviour of the avatars can be interactively updated by changing the map. For example a crowd can be attracted towards a joggler by rendering a darker (ie more dense) polygon using interpolated shading over his neighbourhood. In fact such a density map can almost make the collision height-map redundant since we can set the density over the obstacles to zero and thus preventing any avatar from deciding to go over them.

If we are walking at street level then at any moment the largest part of the static and dynamic scene is not visible. There is a variety of efficient algorithms for culling away the hidden static regions. However visibility culling with such a number of dynamic entities is very much an open problem. Extending one of the cell visibility algorithms (eg [5]) seems like a good avenue to follow. Lets assume we are using the density map described above. At pre-processing the scene and the map are hierarchically subdivided into cells. When the viewer first moves into a region, the culling algorithm is applied which outputs the set of cells which are potentially visible (PVC) while the viewer stays within the region. We can restrict the simulation to only the avatars within the PVC's. However this will make the number of visible avatars drop to potentially zero if we stay in the same view-region long enough, since some avatars will be moving out of the PVC's but none will be coming in. To eliminate this problem we can use the density information. During the culling (which happens only once every several frames) we can identify the edges of the visible cells which form the boundary between visible and non-visible. There we can define 'avatar generators' process which will sample the densities on either side and periodically send in some avatars.

References

1. A. Aubel, R. Boulic, and D. Thalmann. Lowering the cost of virtual human rendering with structured animated impostors. In *Proceedings of WSCG 99*, Plzen, Czech Republic, 1999.
2. Ucl bartlett school of architecture, pedestrian movement project. http://www.bartlett.ucl.ac.uk/spacesyntax/pedestrian/pedestrian.html.
3. Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243–254, 1998. ISSN 1067-7055.
4. Collaborative virtual environments project. http://coven.lancs.ac.uk/.
5. Frédo Durand, George Drettakis, Joëlle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. *To appear in the proceedings of SIGGRAPH 2000*, 2000.
6. Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In Pat Hanrahan and Jim Winget, editors, *ACM Computer Graphics (Symp. on Interactive 3D Graphics)*, pages 95–102. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.
7. Karol Myszkowski, Oleg G. Okunev, and Tosiyasu L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–512, 1995. ISSN 0178-2789.
8. Jarek Rossignac, Abe Megahed, and Bengt-Olaf Schneider. Interactive inspection of solids: Cross-sections and interferences. *Computer Graphics*, 26(2):353–360, July 1992.
9. Germot Schaufler and Wolfgang Sturzlinger. A three-dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):C227–C235, C471–C472, September 1996.
10. Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–

82. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

11. François Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulationfor real-time visualization of urban scenery. *Computer Graphics Forum*, 16(3):207–218, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055.
12. D.Thalmann S.R. Musse, F. Garat. Guiding and interacting with virtual crowds in real-time. In *Proceedings of Eurographics Workshop on Animation and Simulation*, pages 23–34, Milan, Italy, 1999.
13. F. Tecchia and Y.Chrysanthou. Real-time visualisation of densely populated urban environments: a simple and fast algorithm for collision detection. In *Eurographics UK*, April 2000. to appear.
14. Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 43–50. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

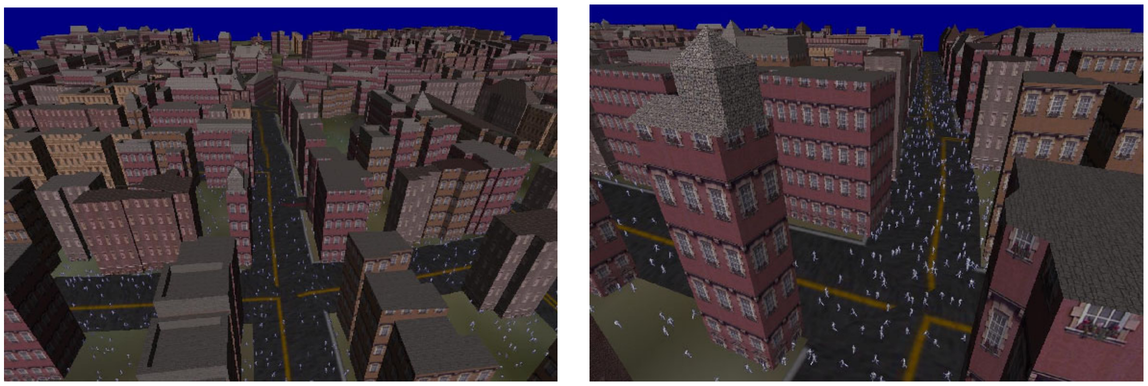


Fig. 5. Views from a distance, of central London populated with 10,000 humans

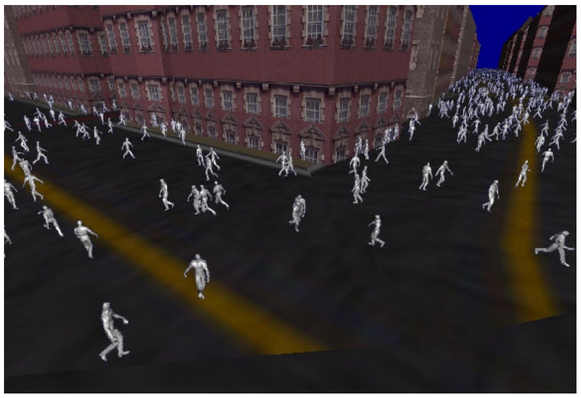


Fig. 6. A view from closer down at Regent Street

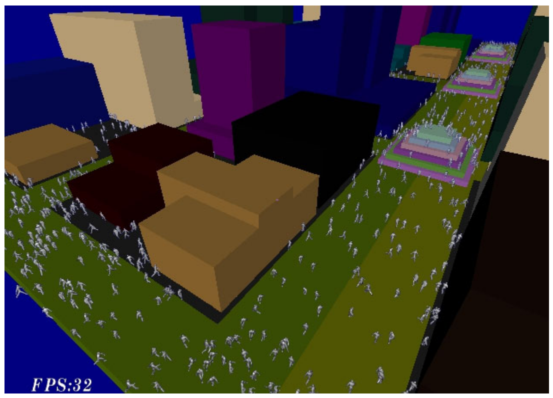


Fig. 7. The climbing of the stairs is done using only information from the height-map