Image-Based Crowd Rendering

Franco Tecchia and Céline Loscos
University College London

Yiorgos Chrysanthou
University of Cyprus

We propose methods for rendering real-time animated crowds in virtual cities. We developed an image-based rendering approach for displaying multiple avatars.

Populated virtual urban environments are important in many applications, from urban planning to entertainment. At the current stage of technology, users can interactively navigate through complex, polygon-based scenes rendered with sophisticated lighting effects and high-quality antialiasing techniques. As a result, animated characters (or agents) that users can interact with are also becoming increasingly common. However, rendering crowded scenes with thousands of different animated virtual people in real time is still challenging.

To address this, we developed an image-based rendering approach for displaying multiple avatars. We take advantage of the properties of the urban environment and the way a viewer and the avatars move within it to produce fast rendering, based on positional and directional discretization. To display many different individual people at interactive frame rates, we combined texture compression with multipass rendering.

Our system allows real-time rendering of densely populated large-scale environments. The rendering speed is independent of the avatar model's complexity, although rendering the same number of humans would be impossible if using polygonal models. We improved already existing methods by three main contributions. (See the "Related Work" sidebar for more information on other approaches.) First, we adapted the choice of the impostor to the object to render, thus minimizing popping effects when changing view. Second, we reduced the amount of texture memory, letting us load many different kinds of people. Finally, we used a multipass algorithm, taking advantage of the alpha channel to select and color different regions of the body, which enabled the 10,000 people simulated in our experiment to all look different. With the continuous increase of the texture memory available in common machines, we predict that using such image-based rendering approaches will provide solutions to crowd visualization.

Approach in a nutshell

Rendering populated urban environments requires the synthesis of two separate problems: real-time visualization of large-scale static environments and visualization of animated crowds and traffic. Because both are expensive to render, it's essential to reduce the amount of time required to display each frame. We believe that our effort on real-time animated crowd display, combined with accelerating techniques for walk-throughs in virtual environments, should allow high-quality visualization of big cities.

Large-scale environments contain millions of polygons. Although we can display and visualize thousands of polygons at a real-time frame rate, delays appear between frames for a larger number of polygons, decreasing the visualization's quality and the user's ability to walk through. There has been a lot of published work on this subject. In general, we can use three different classes of methods to accelerate rendering of large environments: visibility culling, imaged-based rendering, and level-of-detail representation. In our case, we need to reduce the number of polygons to display and take care of the avatars' real-time animation. Visibility culling can be an efficient acceleration in urban scenes, but we would still need to render many polygons. For example, think about a crowded square. A user might visualize thousands of virtual avatars and view the surrounding city details. Even the additional use of level-of-detail techniques results in too many polygons to display.

Considering these limitations, an image-based approach seemed more suitable for both the animation and the rendering of the avatars. We focused on the lowest level, when the viewer is at a certain distance from the virtual humans and potentially has many of them in view. In applications where users need to have a closer look, we can render only the close avatars with polygons.

To minimize geometrical complexity, we represent each human with a single adaptive impostor. (See the "Model Sampling and Collision Avoidance" sidebar on p. 38 for more details.) We selected appropriate impostor images depending on the viewpoint position and the animation frame. A previous approach[1] has already proposed such a solution. However in that approach, the required texture memory is excessive, resulting in a sim-

0272-1716/02/$17.00 © 2002 IEEE

Related Work

The principle of image-based rendering techniques is to replace parts of a scene's polygonal content with images. We can either compute these images dynamically or a priori. Maciel and Shirley[1] used prerender images to replace polygonal parts of a static environment in a walk-through application. They do this individually for single objects or hierarchically for clusters of objects. These images are used in a load-balancing system to replace geometry, which is sufficiently far away. A year later Schaufler and Sturzlinger[2] and Shade et al.[3] independently presented the concept of dynamically generated impostors. They generate object images at runtime and reused them for as long as the introduced error remained below a threshold.

We can find numerous algorithms in the literature that try to generate better approximations. For example Chen and Williams;[4] Debevec, Taylor, and Malike;[5] and McMillan and Bishop[6] warp the images to adapt them to different viewpoints while Mark, McMillan, and Bishop[7] and Darsa, Silva, and Varshney[8] apply the images on triangular meshes to better approximate the object's shape. Schaufler[9] proposes a hardware-assisted approach to image warping, and Dally et al.[10] use an algorithm that efficiently stores image data starting from a number of input images.

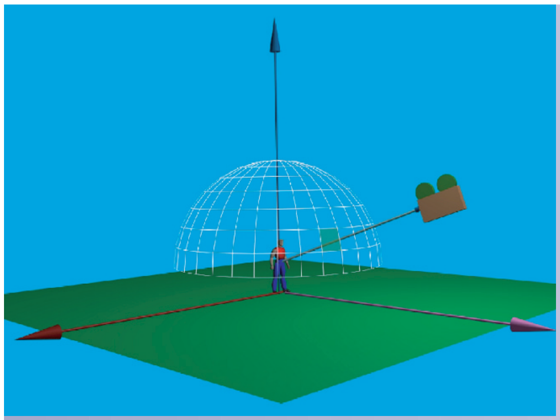
The literature on human modeling and rendering is also extensive. However, the largest part of it concerns achieving realistic approximations using complex and expensive geometric representations. Even with the help of level-of-detail techniques, it would be almost impossible to use a large number of such representations in a real-time system. An alternative, which Aubel and his colleagues[11,12] recently employed, is using impostors to render virtual humans. In one work,[11] they replace each human with a single impostor, while in another,[12] they replace each body part with an impostor, overall using 16 impostors for each human. In both methods, they compute the impostors dynamically and use them only for a few frames before discarding them.

Tecchia and Chrysanthou[13] propose a less accurate but more scalable method that uses fully precomputed images. They show results with only one individual replicated many times due to the excessive texture requirements. Part of our work is based on this approach, which mapped an appropriate texture onto an impostor to display walking humans. To generate the impostors, Tecchia and Chrysanthou created a set of textures, each corresponding to an animation frame. Each texture is composed of a set of images of the character taken from different positions. They used a sampled hemisphere to capture the images, from 32 positions around the character and eight elevations. At runtime, depending on the view position with respect to each individual, they chose the most appropriate image and displayed it on an impostor, which is a single polygon dynamically oriented toward the viewpoint. No interpolation is used between views, as this would be too CPU-intensive. The appropriate texture to map depends on the viewpoint and animation frame. To improve the rendering speed, they draw the humans frame by frame of animation and load the textures only once per frame of rendering.

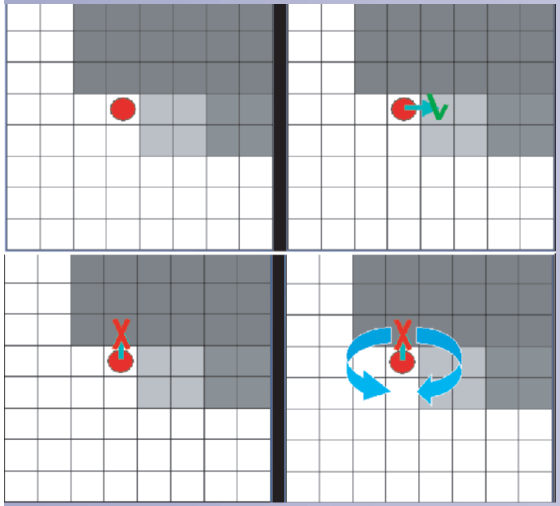
However, several limitations exist in the technique. First, it needs a lot of texture memory because 32×8 sample images need to be stored in one texture. We can also see popping between frames of animation due to the sampling of the view position—there are 11.25 degrees of difference in the orientation of the object between each image. Because of the texture memory's cost, the authors showed animation only for a single type of character. We decided to use this method as a basis for his high efficiency, and we tried to reduce memory requirements to the minimum.

References

1. P.W.C. Maciel and P. Shirley, "Visual Navigation of Large Environments Using Textured Clusters," P. Hanrahan and J. Winget, eds., *ACM Computer Graphics* (Symp. Interactive 3D Graphics), 1995, pp. 95-102.
2. G. Schaufler and W. Sturzlinger, "A Three-Dimensional Image Cache for Virtual Reality," *Computer Graphics Forum*, vol. 15, no. 3, Sept. 1996, pp. C227-C235.
3. J. Shade et al., "Hierarchical Image Caching for Accelerated Walk-throughs of Complex Environments," H. Rushmeier, ed., *Proc. Siggraph 96*, Addison Wesley, Reading, Mass., 1996, pp. 75-82.
4. S.E. Chen and L. Williams, "View Interpolation for Image Synthesis," J.T. Kajiya, ed., *Computer Graphics* (Siggraph 93 Proc.), vol. 27, ACM Press, New York, 1993, pp. 279-288.
5. P.E. Debevec, C.J. Taylor, and J. Malik, "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach," *Proc. Siggraph 96*, Addison Wesley, Reading, Mass., 1996, pp. 11-20.
6. L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Computer Graphics*, vol. 29, 1995, pp. 39-46.
7. W.R. Mark, L. McMillan, and G. Bishop, "Post-Rendering 3D Warping," M. Cohen and D. Zeltzer, eds., *Proc. 1997 Symp. Interactive 3D Graphics*, ACM Press, New York, 1997, pp. 7-16.
8. L. Darsa, B.C. Silva, and A. Varshney, "Navigating Static Environments Using Image-Space Simplification and Morphing," M. Cohen and D. Zeltzer, eds., *Proc. 1997 Symp. Interactive 3D Graphics* (ACM Siggraph), ACM Press, New York, 1997, pp. 25-34.
9. G. Schaufler, "Per-Object Image Warping With Layered Impostors," *Proc. 9th Eurographics Workshop Rendering 98*, Springer Computer Science, New York, 1998, pp. 145-156.
10. W.J. Dally et al., *The Delta Tree: An Object-Centered Approach to Image-Based Rendering*, tech. memo AIM-1604, Massachusetts Inst. of Tech., Artificial Intelligence Laboratory, May 1996.
11. A. Aubel, R. Boulic, and D. Thalmann, "Animated Impostors for Real-Time Display of Numerous Virtual Humans," J.-C. Heudin, ed., *Proc. 1st Int'l Conf. Virtual Worlds* (VW-98), vol. 1434, Springer, Berlin, 1998, pp. 14-28.
12. A. Aubel, R. Boulic, and D. Thalmann, "Lowering the Cost of Virtual Human Rendering with Structured Animated Impostors," *Proc. 7th Int'l Conf. in Central Europe on Computer Graphics, Visualization, and Interactive Digital Media* (WSCG 99), Univ. of West Bohemia Press, Czech Republic, Plzen, 1999.
13. F. Tecchia and Y. Chrysanthou, "Real-Time Rendering of Densely Populated Urban Environments," *Proc. Rendering Techniques 2000*, Springer Computer Science, New York, 2000, pp. 83-88.



A Sampling the models.



B A simple collision test. Particles compare the elevation of the actual and destination cells.

Model Sampling and Collision Avoidance

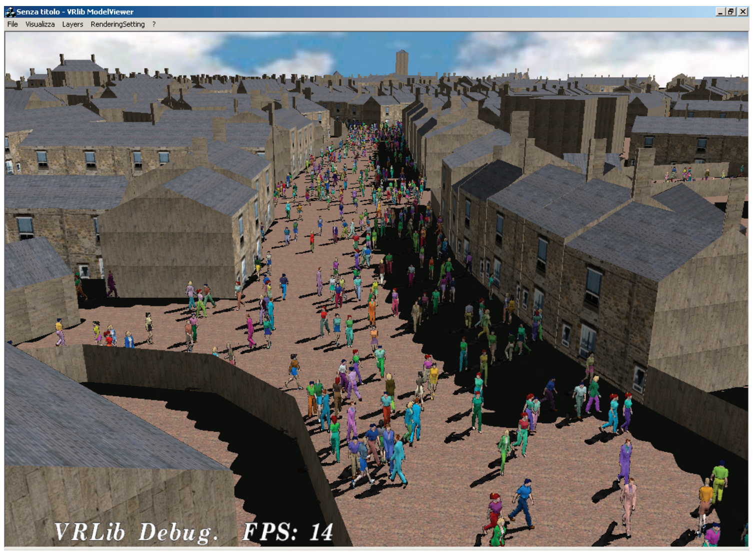
Tecchia and Chrysanthou[1] initially reported the idea of using prerendered animated impostors for real-time crowd rendering. As in that work, we took sample images of the detailed human model from around a discretized hemisphere, using a regular subdivision (32 views across and eight different elevations). We created and animated the original polygonal models using Curious Labs Poser (see Figure A).

To develop some simple behavior simulation, we implemented fast and efficient collision detection using a regular grid.[1] We call this grid a *height map*, with each of its cells representing about 30 × 30 cm of the environment and storing the elevation at that point. We constructed this information efficiently using rasterization hardware.

At start-up, we take an orthographic rendering of the static model with the camera looking down from above. Then the contents of the *z*-buffer are read making the height map (see Figure B). When the humans move around, they access this information to check the environment's properties. We interpret high discontinuities in the elevation of adjacent cells as obstacles. Agents moving around detect such discontinuities and adjust their direction accordingly. (Tecchia et al. also reported a more general use of a 2D grid to control crowd behavior.[2])

ulation that includes only one type of avatar. Our work increases rendering quality using aggressive optimizations and adds important environmental effects such as shadows.

We analyzed all the improvements needed to allow more variety with an increase of the visual quality, while keeping a real-time frame rate. This led us to the set of new techniques we present here, which together display crowds with improved quality while keeping the rendering cost low. We minimize the popping effect when changing views by choosing the impostor representation that best fits walking humans. We can apply the technique we used to select the best-fitting impostor, however, to other kinds of objects. We also decided on a strategy to decrease the amount of texture memory required for one human, as well as find new displaying methods to make every avatar look different. To minimize the memory consumption, we dropped the images' regular-grid organization, removing all the unused space in the impostor images set, which reduces the memory requirements by about three-quarters. This compression technique reduces the size of the required texture memory for each kind of human, letting us add several kinds of humans. To enhance the crowd variety without increasing the memory usage, we used multipass rendering. Figure 1 shows an example of the final rendering system.



1 The crowd rendering system.

Choosing the impostor representation

When we use image-based representation to render complex objects, two common forms of artifacts may arise: missing data due to interocclusion can cause black regions to appear, and popping effects can occur when we warp and/or blend the image samples to obtain the final image. As we already mentioned, we try to maximize the rendering speed using a minimal geometric complexity for each impostor; this leads us to use a single polygon as the plane on which to project a sample. In this scenario, the main perceived artifact is the popping between different samples as the viewpoint changes. Unfortunately, the multipass algorithm our system uses to improve the crowd variety prevents us from blending different samples, a solution that could have mitigated the problem.

Another way to reduce the popping effect could be to augment the number of samples. However, because we still want to minimize the memory consumption, we preferred to use some other methods. Because we have only a limited number of image samples available, we decided to accept this popping effect up to a certain extent, while attempting to minimize it.

The popping artifact occurs because all the points on the surface of the sampled object are projected onto the same plane, from the direction that the camera is facing when the sample is created. Obviously, as the camera position changes, the projection of such points on the impostor can't change, and the current impostor is no longer an exact replica of the object appearance. The amount of error for a generic point on the object surface is proportional to the distance of the point from the projection plane. Figure 2 demonstrates this.

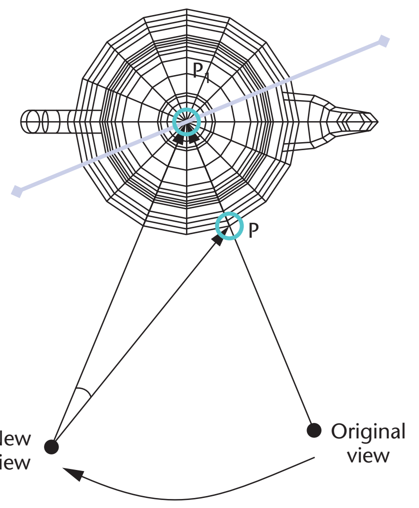
The plane researchers commonly use as the projection plane for an impostor is usually the one perpendicular to the view direction from which the sample image was taken. This plane doesn't take into account the object's shape nor any kind of special occlusion that could be in the image. We then decided to try a different approach: Given an object and the camera position from where we created the sample image, we searched for the projection plane passing through the object that minimizes the sum of the distances of the sampled points and the projection plane.

To apply this idea, we had to project back in the 3D space the points visible in the impostor image to get 3D visible samples of the object. We applied a Principal Component Analysis (PCA) to the set of 3D points and identified the two principal eigenvectors as directions describing the projection plane. In the case of sample human polygonal models, such a plane results in a significantly better approximation of the position of the visible pixels with respect to the actual point positions in 3D. Unfortunately, other visual artifacts arose when we used the best-fitting plane as the impostor plane. In fact, the special orientation of this new plane asymmetrically warps the image depending on which direction the camera moves away from the sampling position. In some extreme cases, for a particular plane orientation and a certain distance of the camera, perspective distortions can also become too evident (see Figure 3a). This is more visible when moving upward rather than around because our object (the avatar) is longer along that dimension.

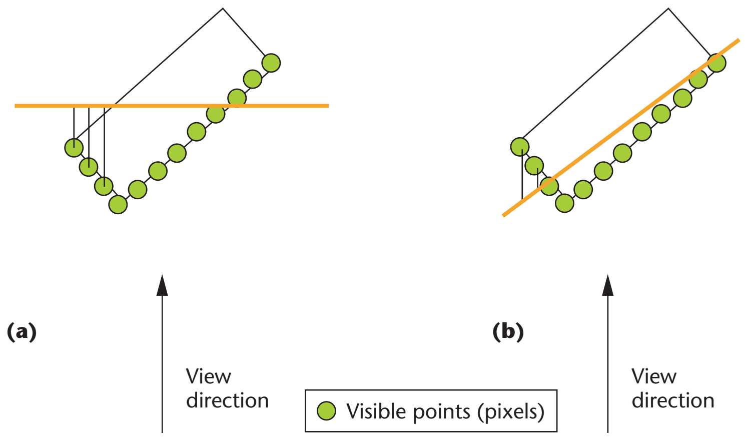
Because we couldn't use the best-fitting plane computed with PCA as is, we reduced the artifacts by combining the plane's two candidate orientations. Starting from an impostor plane purely perpendicular to the camera, we "perturbed" its orientation using the resulting plane obtained from a PCA of the sample image (see Figure 3b). This minimized the popping while limiting the introduction of other artifacts. In practice, we found that averaging the two directions worked well.

Image compression

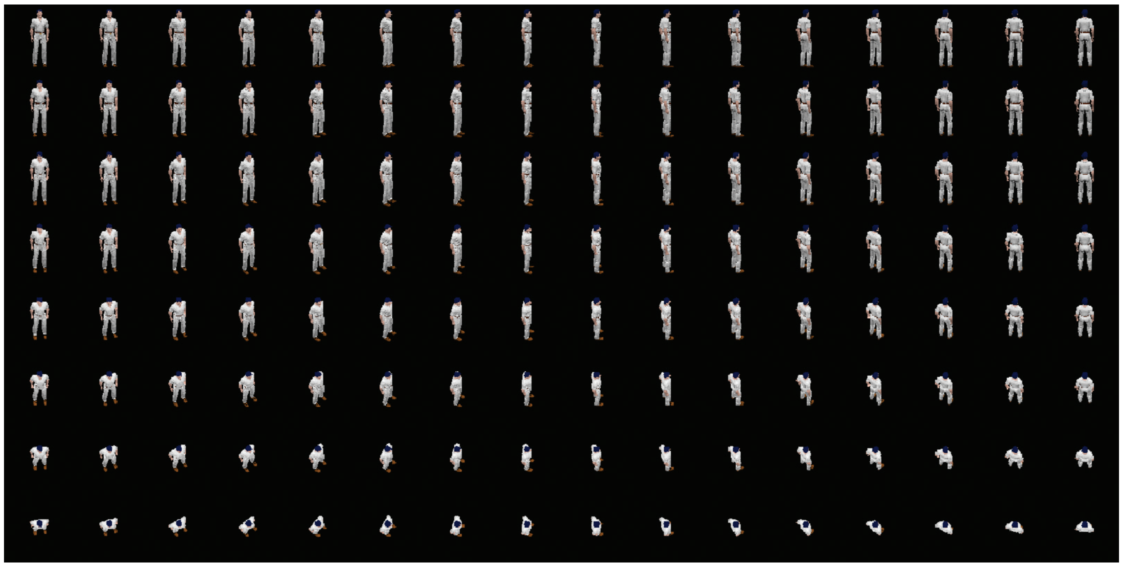
Although the hardware texture memory available has increased, we should still make an effort to reduce the amount used to display virtual humans. First, because humans are walking, the movement is symmetric. Instead of 32 samples, we can then reduce it to 16 and get the other 16 by mirroring the texture. We can see such symmetry in other objects, such as cars or bicycles,



2 Error introduced when changing the viewpoint. This error is proportional to the distance between the 3D point and the projection plane.



3 Distance of the visible samples from the impostor plane. (a) The plane is perpendicular to the camera direction. (b) We chose the best-fitting plane to minimize the distance between sample points and the projection plane.



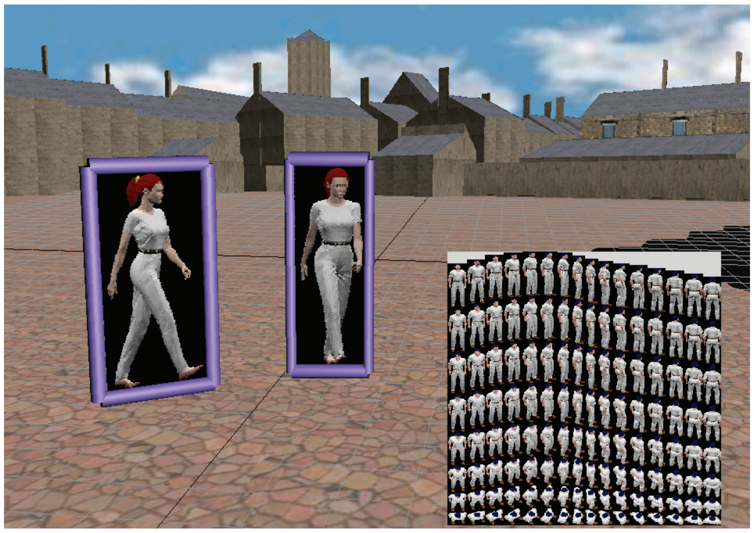
4 Impostor texture for an animation frame using the Tecchia and Chrysanthou method.[1]

and this approximation may be useful for them as well.

Second, all the images are the same size. This results in a considerable waste of space, because for some images, the human fits a restricted area. We started by placing the samples on a texture using a regular grid.[1] Each sample is a prerendered ray-traced image of 256 × 256 pixels of the character, using an orthographic projection. Figure 4 shows the resulting image. In this way, extracting a particular sample is a trivial and fast operation, but there's a lot of unused texture space around each sample that gets wasted. To minimize the amount of unused regions, during the preprocessing phase, we computed the smallest rectangle containing the character for each sample. Then, we combined all these samples in a single image, reorganizing them to minimize the unused space. Thanks to this process, the resulting new image is much smaller than the original without any loss of image quality.

With our current reorganization strategy, we can reduce the amount of texture memory necessary to store our human images to 25 percent of the original value. In our case, to have a good trade-off between the quality and the memory required for the samples, we stored each animation frame using a single image of 512 × 512 pixel as a total size. Figure 5 shows an example of the resulting texture. We then stored the texture using the OpenGL compressed format (http://oss.sgi.com/projects/ogl-sample/registry/EXT/texture compressions3tc.txt), which gives a further memory compression ratio of 1:4. This ratio is efficient, although the image loses part of its quality. The compression format lets us keep alpha values and encodes them in 4 bits. Once loaded in texture memory, each animation frame for a single human model requires 256 Kbytes.

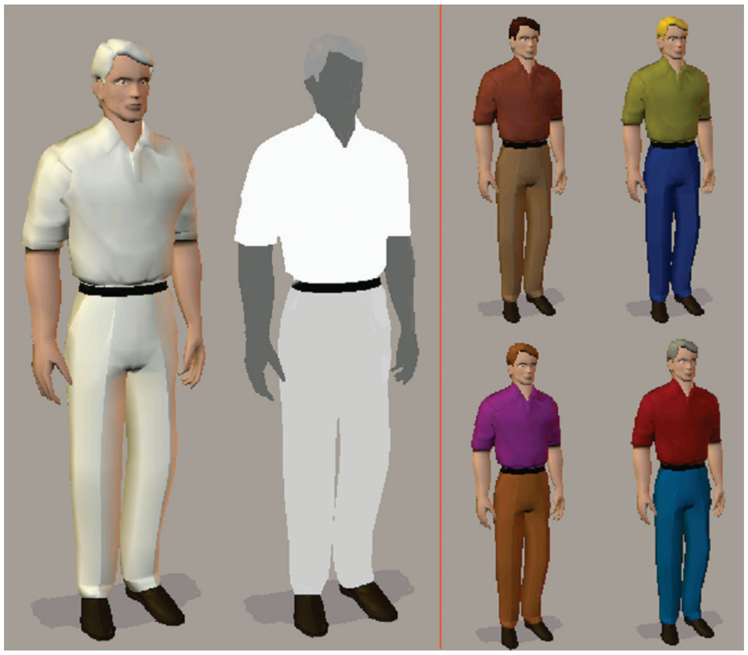
Because we no longer have a regular grid, we now must precompute appropriate texture coordinates and store them for each sample. Then, at rendering time, we need to compute on the fly the right size and orientation for the impostor to avoid introducing distortions of the sample image. It's important to notice that, because of our optimal samples placement strategy, these parameters generally vary for different frames of animation even considering a fixed point of view. Figure 5 shows an example of the mapping and the choice of the impostor.



5 Rendering impostors using a compressed texture. The bottom right shows the texture after compression. We packed the images so they occupy only one fourth of the texture in Figure 4.

Increasing the variety of avatars

With this texture compression we gain texture memory that we can use to simulate more humans than Tecchia and Chrysanthou.[1] To simulate 10 different types of human with 10 frames of animation each, we need about 25 Mbytes of texture memory. Although we improved the possibility of variety, 10 different avatars aren't enough to populate a city. Because we were limited by the texture memory, we decided to modify the texture on the fly using multipass rendering. We can't change the shape and the



6 Example avatars. On the left, an avatar rendered with ray tracing in 3D Studio Max. In the middle, an avatar with alpha channels to identify parts to modify. On the right, an avatar rendered with multipasses regarding the alpha channel. Notice how these four avatars look different although they are built from the same 3D model.

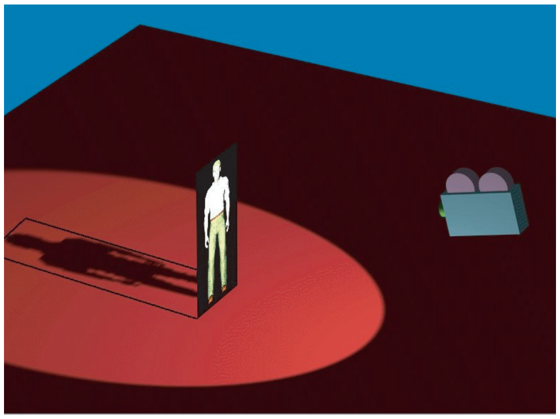
kind of human, but we can assign a different color to significant parts of the body such as clothes, hair, and skin color.

To identify such areas, we precomputed an alpha-channel image with a different alpha value for each part to modify (see Figure 6). (We did this using 3D Studio Max. We had to turn off the antialiasing to avoid blurring the borders of two adjacent regions.) Tuning the alpha channel, we can define up to 256 different regions in the texture without using compression, or up to 16 using the s3tc compression (http://oss.sgi.com/projects/ogl-sample/registry/EXT/texture compressions3tc.txt) because only 4 bits are available for the alpha channel. We performed multipass rendering using the alpha channel to select the parts to render. For each pass, we change the impostor polygon color to the desired color, and we apply the texture using the flag GL_MODULATE and set the alpha threshold of the alpha test to the one associated with the part of interest.

Computing a texture modulation preserves the shading because it's already included in the texture. In our experiments, we drew up to three passes, thus only changing each individual's shirt and trousers color. More passes are possible because we can identify several regions. However, the multipass rendering might slow down the overall rendering rate, and there must be a trade-off between variety and rendering time.

Implementation details and results

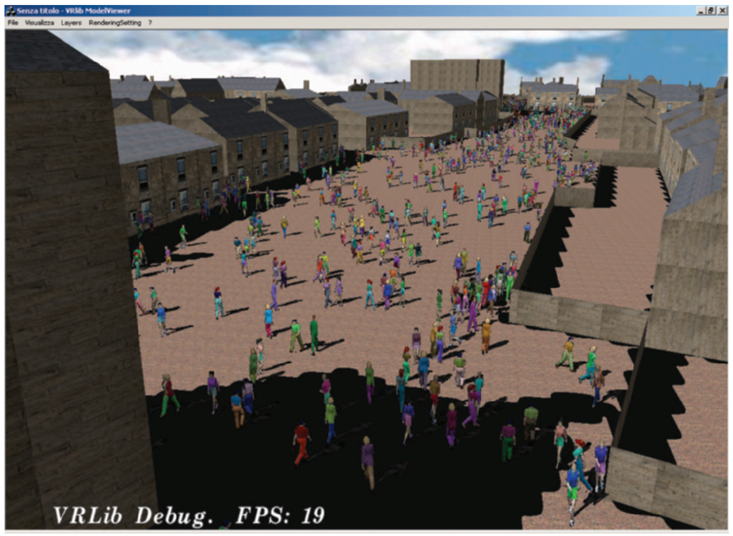
We implemented and tested the methods we discuss here. For the simulation, we added some elements that allow better quality for the results. To control the virtual humans' motion, we subdivided the floor of the environment into regular-sized tiles. While the humans move around, they check information corresponding to the tile they occupy. Several types of information can be stored,[2] but presently, we only use this information for collision detection and shadowing. We further subdivided the tiles so that an avatar can be at different positions on the same tile.



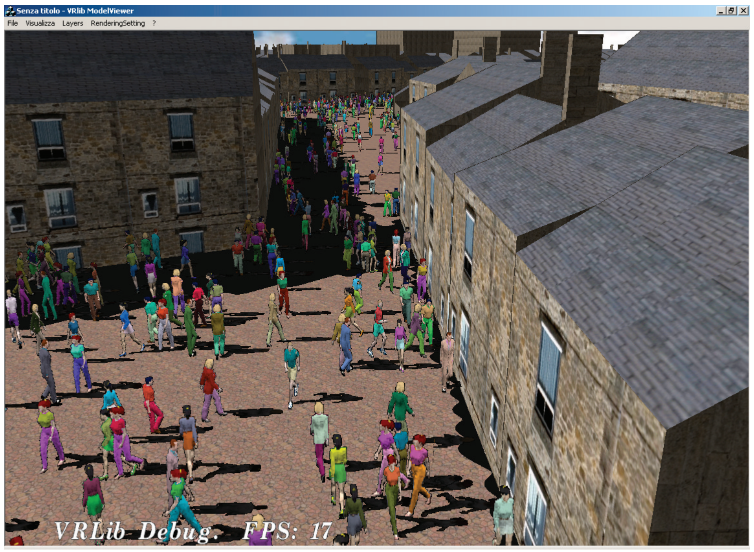
7 A virtual human's shadow is the projection onto the ground of its silhouette as seen from a single light source.

If an avatar stays in the same tile in the next frame, it just continues to move in its current direction and no decision needs to be taken. In the case of the model we used for the test, the collision detection map is binary. When we encode the tile with black, it's impassable and the avatar needs to change direction. In addition, we performed intercollision detection between humans by checking if a destination tile is already occupied.

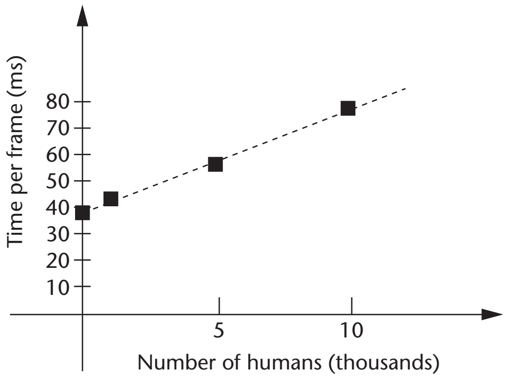
Using the impostor approach, we can also compute and display the moving humans' shadows.[3] To project the humans' shadows on the ground, we exploit the fact that the shadow of a virtual human is the projection onto the ground of its silhouette as seen from the light source (see Figure 7). We can thus display the humans' shadow on the ground using one of the images already stored in the database that we use for the impostor. In this way, with the cost of only one additional polygon corresponding to the projection of the impostor polygon on the ground, we create a shadow corresponding to the human posture.


Senza titolo - VRlib ModelViewer
File Visualizza Layers RenderingSetting ?
VRLib Debug. FPS: 19

8 The crowd visualization. Notice the number of different people. Using the optimization techniques in this article, we visualized thousands of different humans in real time.


Senza titolo - VRlib ModelViewer
File Visualizza Layers RenderingSetting ?
VRLib Debug. FPS: 17

9 A closer view of the humans.



10 Number of humans versus the time per frame.

We also cast the buildings' shadows onto each virtual human extending the function of the regular grid used for collision detection. We compute and store in a 2D image the information about the height of a building's shadow volumes. For each virtual human traversing the grid, we compare its height with the height of the shadow stored in the cell it occupies. The impostor polygon appropriately darkens to reflect the shadow coverage.

We developed the system on an 800-MHz PC Pentium III with an Nvidia GeForce GTS2 video card. We populated our environment with six different avatars and performed three passes to draw different colors, chosen randomly. For the results in Figures 8 and 9, we displayed 2,000 avatars for each of four types of humans and 1,000 for each of the two last type, thus displaying 10,000 different humans. One of these types is a jogger, thus having an animation different from the others. These humans move in a village modeled with 41,260 polygons. The display updates between 12 and 20 frames per second (frames/sec), depending on the displayed polygonal complexity. Although the rendering is in real time, there's no trade-off made to decrease the quality. The rendering quality is as good as if we hadn't used optimization algorithms. Visit http://www.cs.ucl.ac.uk/research/vr/Projects/Crowds/CGA/ for sample videos of this implementation.

To evaluate our simulation's scalability, we simulated different sized crowds. The frame per second rate for 0 people was 26.20; for 1,000 people, it was 24.08; for 5,000 people, it was 18.26; and for 10,000 people, it was 13.35. We ran the simulation on an identical camera path for each simulation.

Rendering the city model itself already uses a lot of the resources because the average display is only 26 frames/sec. In Figure 10, we plotted the same data but as time per frame versus the number of humans, and we can clearly see that the relation is almost linear. We believe that an occlusion culling algorithm performed on the static model could help speed up the rendering. The frame rate then decreases as the number of polygons (those for the avatars) displayed increases. These timings also include the collision detection performed for each of the virtual humans simulated. A visibility test and an occlusion culling algorithm applied to both the collision detection and the display of the humans could accelerate the frame rate.

Future work

Although we've already achieved good results, we believe that there's room for further improvements and developments. In our implementation, we made a number of assumptions that we can now reexamine. For example, we assumed that the viewer will be at a certain distance away from the avatars. This lets us use impostors that we created with orthographic projection and limited texture dimensions. If we let users get closer, they will notice the artifacts. One way around this is to use a hybrid approach where polygonal human models are used instead of impostors for the few avatars that come right up to the viewpoint.

Because we used the same textures to generate the avatar shadows, we also assumed that the light source at infinite. This wasn't a limitation for our examples because the only light source was the sun. However, if we want to simulate the city at night with street lights, then we will have to warp the textures before applying them.

The impostors are currently shaded when we render them in 3D Studio Max. This effectively fixes the shading to the particular light defined at that time, making the sources static. However, we believe that we could shade the impostors on the fly using normal maps indicating each pixel's orientation. This would let the shading be consistent if we modified the light source.

An interesting extension to our system, that could greatly improve its impact, would be using real photographic images of humans instead of synthetic models. Of course, the problem here would be acquiring all the sample images. Possibly, we could scan a person in color and then take the samples from the scan.

Methods to cull polygons could speed up rendering times, selecting for display both polygons from the static environments and the moving people.[4]

We could certainly use the current algorithms to render different kinds of objects such as cars, pets, children, or groups of people (such as children holding an adult's hands). Also, several animations could be possible, with an appropriate texture load. Particular care should be taken for transitions in between animations. Moreover, using the multipass rendering algorithm, we could simulate simple animation such as turning a head to the left or right.

Finally, for each new type of object, we should do more work on developing appropriate behavior. Although researchers have done a lot of work on behavior in cities, there are still many problems to solve, especially for real-time simulation of thousands of agents. ■

Acknowledgments

This work was in part supported by the UK Engineering and Physical Sciences Research Council (EPSRC) project GR/R01576/01 and the EPSRC Interdisciplinary Research Centre Equator, an interdisciplinary research project at University College London. (See http://www.cs.ucl.ac.uk/research/equator/ for more information.)

References

1. F. Tecchia and Y. Chrysanthou, "Real-Time Rendering of Densely Populated Urban Environments," *Proc. Rendering Techniques 2000*, Springer Computer Science, New York, 2000, pp. 83-88.
2. F. Tecchia et al., "Agent Behaviour Simulator (abs): A Platform for Urban Behaviour Development," *Proc. Game Technology* (GTEC 2001), CD-ROM, 2001.
3. C. Loscos, F. Tecchia, and Y. Chrysanthou, "Real-Time Shadows for Animated Crowds in Virtual Cities," *Proc. ACM Symp. Virtual Reality Software and Technology*, ACM Press, New York, 2002, pp. 85-92.
4. F. Tecchia, C. Loscos, and Y. Chrysanthou, "Real Time Rendering of Populated Urban Environments," *ACM Siggraph Technical Sketch*, ACM Press, New York, Aug. 2001.

Franco Tecchia *is a research fellow at the University College London, working on real-time rendering of populated complex environments. His research interests include real-time computer graphics, virtual and augmented reality, and software engineering. He received his DrEng degree in computer engineering from the University of Pisa.*

Céline Loscos *is a lecturer at the University College of London. Her research interests include inverse illumination and realistic illumination for complex environments. She participates in European Union and other funded projects for real-time rendering for mixed interfaces, such as real-world data or haptics. She received an MSc and a PhD in computer science from the Université Joseph Fourier in Grenoble, France. She is an ACM and Eurographics member.*

Yiorgos Chrysanthou *is an assistant professor at the University of Cyprus. His research interests are in computer graphics, virtual reality, and computational geometry. He received a BSc in computer science and statistics and a PhD in computer graphics from Queen Mary and Westfield College. He is an IEEE, ACM, and Eurographics member. He also coauthored the book* Computer Graphics and Virtual Environments: From Realism to Real Time *(Addison Wesley, 2001).*

Readers may contact Franco Tecchia at the Dept. of Computer Science, Univ. College London, Gower St., WC1E 6BT, London, UK, email f.tecchia@cs.ucl.ac.uk.