(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 February 2003 (27.02.2003)

PCT

(10) International Publication Number
WO 03/017204 A2

(51) International Patent Classification[7]: G06T 15/10

(21) International Application Number: PCT/GB02/03820

(22) International Filing Date: 19 August 2002 (19.08.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0120158.1 17 August 2001 (17.08.2001) GB

(71) Applicants and
(72) Inventors: **BERGAMASCO, Massimo** [IT/IT]; Via Don Minzoni, 144, Castelmaggiore, Calci (P1), Pisa (IT). **TECCHIA, Franco** [IT/GB]; 41 Mowbray Road Flat C, London NW6 70S (GB).

(74) Agent: **JACOB, Reuben, Ellis**; Edward Evans Barker, Clifford's Inn, Fetter Lane, London EC4A 1BZ (GB).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declaration under Rule 4.17:
— *as to the identity of the inventor (Rule 4.17(i)) for all designations*

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 03/017204 A2

(54) Title: A RENDERING SYSTEM

(57) Abstract: The invention relates to a system for rendering a multiplicity of dynamic three-dimensional objects in a virtual environment, the system comprising a memory for storing a plurality of sample images representing the objects viewed from different angles, the system further comprising processing means configured for: for each object to be displayed, selecting a sample image from the memory based on the angle of viewing the object, wherein the pixels of the sample image include an alpha channel value for identifying a plurality of different regions of the image; mapping the sample image to a polygon representing the location of the object in the environment, including performing multi-pass rendering of the sample image, wherein each pass is for rendering differently coloured regions of the sample image identified by the alpha channel values.

A RENDERING SYSTEM

This invention relates to a system for rendering a multiplicity of dynamic three-dimensional objects in a virtual environment.

The particular embodiments described relate to a simulation of a complex virtual urban environment containing hundreds or thousands of dynamic animated objects (such as virtual humans, vehicles, animals, etc).

Recently, a lot of research has gone into geometrical modelling of three-dimensional environments. A common example of very complex scenarios are cities and other urban environments which, especially in recent years, have been widely used in virtual reality and virtual presence experiments [4]. In particular, it is often desirable to model urban environments, such as towns or cities, for use in a large number of potential applications, such as urban planning, architectural or cultural applications, acclimation using virtual environments and travel rehearsals. Computer generated environments are also widely used for computer games or other entertainment.

At the current stage of technology, virtual environments are often displayed using complex, polygon-based scenes rendered with sophisticated lighting effects and high quality anti-aliasing techniques. Virtual environments through which a user can interactively navigate are also becoming more and more common. A problem common to much of this modelling is how to display in real time the large amount of information contained in the models. Many different computer graphics techniques have been used to efficiently handle the complex

geometric models, up to the point where quite substantial models can now be displayed in real time. However, the real time visualisation of virtual environments which include a large number of moving agents (virtual humans, vehicles, etc.) causes particular problems.

Much research has been carried out into analysing the issue of populating empty models with a crowd of intelligent virtual agents. Examples of this can be found in Tecchia [1], Kallman [5], Farenc [6], Friedman [7], Donikian [8], Donikian and Rutten [9], Donikian and Curzot [10]. In some of these works much effort is devoted to modelling realistic behaviours of the agents so that it is possible to simulate behaviours such as flow of crowds or traffic.

The visualisation of populated virtual environments can often be considered as two separate problems, namely the visualisation of the static environment (such as buildings) and the visualisation of a non-static population (dynamic agents such as virtual people, animals, vehicles, etc.). Both of these are expensive to render, and accordingly it is always important to reduce as much as possible the amount of time required for each frame to be displayed. In particular, problems are experienced when displaying highly populated large-scale environments. A large-scale environment might contain, for example, millions of polygons. This, combined with the display of thousands of non-static agents has not yet been satisfactorily displayed in real time. A trade off is often required between the visual quality and the frame rate of the display.

Much research has previously been carried out into accelerating the rendering of complex environments. In

general, there are three different methods which may be used to speed up the rendering:

Visibility culling [22],

image-based rendering [23,24] and

level-of-detail representations

Level-of-detail and visibility culling can be efficient, but still leave too many polygons to be rendered. In this application, it is necessary to improve the rendering efficiency while also dealing satisfactorily with the animation of the dynamic agents. Accordingly, an image-based rendering approach appears to be the most suitable.

The principle of image based rendering techniques is to replace parts of the polygonal content of the scene with images, mapped onto "impostors". An impostor takes the place of the parts of the scene being replaced. The images are usually mapped onto impostor polygons which define the orientation and location of a plane in the virtual environment onto which the image (also known as texture) is applied. The term "impostor" is also used to mean the combination of the polygon and the image (i.e. a textured polygon).

The images can either be computed dynamically or *a priori*. Maciel [23] used pre-rendered images to replace polygonal parts of a static environment in a walkthrough application. This was done individually for single objects or hierarchically for clusters of objects. These images were used in a load balancing system to replace geometry which is sufficiently far away.

Shaufler [25] and Shade [26] introduced the concept of dynamically generated impostors. In this case object images

were generated during run time and re-used for as long as the introduced error remains below a threshold. Numerous algorithms have subsequently been proposed in the literature, that tried to generate better approximations. For example Chen [27], Debevec [28] and Macmillan [29] warp the images to adapt them to different viewpoints, while Mark [30] and Darsa [31] apply the images on triangular meshes to approximate the shape of the object. Schaufler [32] proposed a hardware assisted approach to image warping and Dally [33] used an algorithm that is very efficient in storing image data starting from a number of input images.

The literature regarding previous approaches for rendering virtual humans is particularly extensive. However, a great majority of this is concerned with achieving realistic approximations with complex and computationally expensive geometric representations. Even with the help of level-of-detail techniques it would be almost impossible to use a large number of such geometric representations in a real time system. An alternative recently employed by Aubel et al [34,35] used impostors for the rendering of the virtual humans. In [34] each human was replaced by a single impostor while in [35] the author took a much more detailed approach where individual body parts are each replaced by an impostor, using sixteen impostors for each virtual human. In both of these methods impostors are computed dynamically and only used for a few frames before being discarded.

One of the main difficulties in the rendering process for a multiplicity of dynamic agents is the very large number of independent elements which must be visualised and animated. It is also necessary to simulate behaviour of the dynamic agents. For example, in the case of virtual humans in a

crowd, the system should ensure that collisions of the virtual humans both with each other and also with the static environment are prevented, and that a realistic behaviour is achieved. However, this application concentrates only on the rendering process and not on the behaviour simulation.

As previously mentioned, Tecchia [1] described a system using image based rendering techniques, which is less accurate than some other techniques but much more scalable. Tecchia used images of human characters which were constructed prior to run time. During run time, one image was applied to each single polygon impostor representing a virtual human. The images showed a human character from various different angles and elevations, and the appropriate image was selected depending on the orientation of the virtual human and the camera position from which the scene is viewed. In this way it is possible to show a good approximation of a human character when viewed from any different angle.

The rendering system disclosed by Tecchia [1] is now described in more detail, with reference to the schematic shown in Fig. 1. Fig. 1 shows a rendering system 100 organised into two distinct phases, a preprocessing phase 110 and a run time phase 111. The preprocessing phase 110, is used to build and store sampled images in an image database 103. An example set of sample images is shown in Fig. 3. Later, during run time an appropriate sample image will be retrieved from the image database 103 and correctly sized and pasted onto a single polygon to display a virtual human.

In order to build the image database 103, non-uniform discretization (step 101) is carried out in which a pre-computed set of discrete angles for viewing an object are

used to determine the particular angles at which image sampling (step 102) will occur. Image sampling is then carried out to obtain views of the object from a number of different directions, either using computer generated modelling of an object, or any other suitable method. In Tecchia, the set of images were obtained using a sampled hemisphere around a three-dimensional modelled human character as shown in Fig. 2. The set of sample images is then stored in the image database in the dedicated texture memory of the system.

A set of 16x8 sample images is shown in Fig. 3, each row representing sixteen different viewpoints around an object (human character) at a certain height (i.e.rotating around the y-axis). Mirroring these views is possible due to the nature of human characters, and gives us 32 different views at intervals of 11.25° around the object at a particular elevation. Each row corresponds to a different elevation of the viewpoint.

In order to obtain an animated object, such as, for example a human character walking or jogging, the images are sampled while the object is performing various stages of the animation. In this case, each set of sample images represents one frame of animation. Since Tecchia [1] was concerned with the display of crowds, the animation chosen was composed of ten different frames of a walking human.

In Tecchia [1] the sample images were produced using two commercially available packages: Kinetix 3D StudioMax (3dmax) and MetaCreations Poser. To build each individual image, a polygonal version of a human character was rendered from a sample view direction and the resulting image stored

in the texture memory. Alpha values were used for pixels representing transparent zones. Since the highest precision was not required, the images were stored with only sixteen bits per pixel - five for each of Red, Green and Blue and one for the alpha value. In this way 1 Mb of uncompressed texture memory was required for each frame.

During the run time phase 111, the camera position relative to the orientation of the agent is compared with the precomputed discrete set of view directions (in the non-uniform discretization step 106) and the closest view direction selected. The impostor selection step 105 then determines the correct orientation and location in 3D space of the impostor polygon. In step 104, the correct image is selected depending on the chosen view direction, and the sample image is retrieved from the texture memory. Finally, the selected sample image is mapped to the impostor for display 107.

The system was implemented using a low-end Intel workstation equipped with a 350 MHz Pentium II processor, 256 Mb RAM and an openGL compliant videocard with 32 MB video memory. In order to maximise the rendering efficiency on this kind of architecture, sample images for a single frame of animation were stored together as shown in Fig. 3. Then, to limit the number of texture changes during the simulation, agents having the same frame of animation were grouped so they could all be drawn together, and thus the relevant portion of the texture memory only had to be loaded once.

The results are illustrated in Figs 4 and 5. Fig. 4 shows how the rendering time required per frame increased with the number of agents (individuals). The bottom curve shows the

rendering time for the static model (e.g. the virtual city) plus an impostor polygon for each dynamic agent (but no texturing, i.e. the sample images have not been mapped to the impostor polygons). In the middle curve, a collision detection algorithm has been added to control the behaviour of the dynamic agents (not discussed in this application). In the top curve, both the collision detection algorithm and textured impostors (i.e. including mapping sample images onto the polygons) are turned on. A small city model of 2368 polygon was used. This graph illustrates the scalability of this method with an increasing number of agents, indicated by the fact that the curves are almost linear. For a particular example, using a model of 2368 polygons, a full collision detection algorithm and 10,000 textured impostors, the system ran at 25 frames per second.

Fig. 5 shows a corresponding graph of how the rendering time per frame increases with the complexity (i.e. number of polygons) of the static environment, namely the model of the city with a fixed number (10,000) of dynamic agents.

However, several limitations can be observed in the system used in Tecchia. Firstly, there is a trade-off between memory usage and the visual quality of the display. For good visual quality it is necessary to ensure a sufficiently large set of sample images; however, increasing the amount of texture memory used significantly reduces the speed of the rendering. A large amount of texture memory was required to store a set of 16 x 8 sample images. In one particular implementation, each sample image had a resolution of 64 x 64 pixels (this was found to be a good compromise in terms of image quality and memory requirements). Accordingly, a texture size of 1024 x 512 pixels was needed to hold all 16 x

8 views. As a result of this texture memory requirement Tecchia[1] decided to display only one type of agent. As a result, although it was possible to display thousands of moving agents in real time, the agents were all identical.

A further disadvantage of Tecchia was the visual artefact of "popping" could be observed when a viewers attention was concentrated on a particular agent. This is when an image appears to jump between two successive frames of the animation. This occurs when the viewing angle changes, and is caused by the discrete nature of the angles chosen for the sample images (there is 11.25° difference between the orientation of the object in subsequent images). Interpolation between images can reduce this effect, but is generally not desirable since it is highly processing-time-intensive.

Finally, in Tecchia [1] the dynamic agents were displayed without shadows, an aspect which is particularly desirable when displaying virtual environments. Accordingly, the problems and increase in rendering time associated with generating and displaying shadows, either shadows cast by the moving agents themselves onto the static environment (e.g. the ground) or the shadowing of parts of the agents themselves as they move in and out of areas of shadow, were not addressed.

In general, in order to achieve fast frame rates for virtual animation, shadowing has often not been included in displays. However, at an early stage in computer graphics, shadows were acknowledged as being very important in both contributing to the realism of virtual environments, and also helping the user to situate objects relative to each other. Since

shadows are so natural for the human eye, their presence greatly improves the overall perceived realism even if other aspects of the virtual environment are not rendered in a particularly realistic way. Typically, shadows are not implemented because prior art shadow computation techniques are too computationally intensive to be performed in real time on complex scenarios, particularly where there may be thousands of dynamic agents.

A large number of techniques exist in prior literature relating to shadow generation, which can be broadly sub-divided into two categories. Soft shadows are produced by assuming a light source is a finite area or volume, while sharp shadows are produced by assuming light is coming from an infinite distance away or from a single point. Soft shadows are much more accurate and realistic, but at the same time they require complex and expensive computation, which makes them not really applicable to real time rendering for complex dynamic virtual environments. Techniques such as discontinuity meshing are the most accurate, as they find not only the shadow boundaries but also edges where the direct illumination on the surfaces varies discontinuously [11, 12]. However, these techniques are slow to compute and render because of the complex mesh. Techniques which use texture mapping instead of partitioning the input polygons into a mesh are faster to render [13, 14]. However, these methods cannot be used to compute shadows for any complex environment in real time, and they require separate textures for different surfaces.

For real time applications, the rendering of shadows has often been restricted to sharp shadows. The simplest method is the fake shadows technique [15]. After rendering a model

from a specific viewpoint a matrix is added on the stack that has the property that it projects every vertex onto a plane (usually the ground). The scene is rendered again with the projected objects as "grey", appearing as shadows. An additional rendering of the scene is needed for every receiving surface. Shadows are computed at each frame.

Some methods [16] pre-compute shadows so they only need to be rendered for each new viewpoint. The computation of the full solution on surfaces is possible but is time consuming. Incremental updating is possible when there is a small number of dynamic agents [17], but this would be too costly for a highly complex dynamic virtual environments. In addition, even ignoring the updating cost, the number of additional polygons needed for the display of the shadows can greatly affect the rendering performance.

Currently, the most popular techniques use graphics hardware to compute the shadows at a per pixel level, thus avoiding the shadow storage problem.

Shadow volumes methods [18] were used to delimit a spacial region in shadow between the object surface and a receiver. Using a stencil buffer [19], regions in shadow can be automatically by the hardware. An interesting alternative method for computing shadow planes was suggested by McCool [20] a scene is first drawn from the light position and the z-buffer read. The shadow volumes are then reconstructed based on the edges of the discretised shadow map. However, the discretisation can introduce undesirable artefacts.

One of the drawbacks of this method is that the shadow planes tend to be very large and they have a detrimental affect on

the rendering time. In practice, this effect can be limited by using the method not for a complete shadow solution, but rather only for shadows for "important" objects in the scene. Similar to the shadow Z-buffer this method can be used to cast shadows onto any object that can be scan converted. But unlike the shadow Z-buffer the object causing the shadows need to be polygonal.

Methods based on shadow maps [21] are pixel based, mixing depth information provided both from the light source and the user viewpoint. Shadow areas are determined by comparing the depth of the points from both the light source angle and the user viewpoint. Such methods are now implemented in hardware allowing fast shadow computation for complex geometric virtual environments. However, these methods suffer from two major drawbacks due to image resolution. Firstly, images can be aliased if the resolution does not permit an accurate determination of depth. In particular, this is often the case in complex scenes where distant objects are represented by only a few pixels. Secondly, because the method is pixel based, the frame rate depends on the size of the display image.

It is an object of the present invention to seek to mitigate the disadvantages of the prior art.

According to a first aspect of the present invention, there is provided a system for rendering a multiplicity of dynamic three-dimensional objects in a virtual environment, the system comprising a memory for storing a plurality of sample images representing the objects viewed from different angles, the system further comprising processing means configured for: for each object to be displayed, selecting a sample

image from the memory based on the angle of viewing the object, wherein the pixels of the sample image include an alpha channel value for identifying a plurality of different regions of the image; mapping the sample image to a polygon representing the location of the object in the environment, including performing multi-pass rendering of the sample image, wherein each pass is for rendering differently coloured regions of the sample image identified by the alpha channel values.

The processing means may be further configured to store in the memory in relation to each type of object at least one set of sample images representing one of the objects viewed from a set of discrete viewing directions. This allows the correct angle of viewing of the object to be easily selected when the appropriate portion of texture memory is loaded.

The processing means may be further configured to store in the memory in relation to each type of object a plurality of sets of sample images, the sets corresponding to different frames of animation of the object. This advantageously allows any objects having the same animation frame to be grouped together for rendering such that the relevant portion of texture memory only has to be loaded once.

The processing means may be further configured to perform image compression on an original set of sampled images before storing the sample images in the memory, wherein the original sample images each comprise a background portion and an object portion, the compression comprises: revising each of the sample images by discarding part of the background portion which is not enclosed by a smallest-fit rectangle around the object portion; and reorganising the arrangement

of storing the revised sample images to minimize unused storage space in the memory. This advantageously allows the amount of texture memory required to store the sample images to be reduced.

The processing means may be further configured to: select one of a plurality of projection planes stored in the memory based on the angle of viewing the object; and map the selected sample image onto the selected projection plane. This allows the system to conveniently select the most appropriate projection plane for the sample image, to reduce unwanted visible artefacts such as "popping" during the simulation.

The processing means may be further configured to: for each sample image, compute a plane for said image and store the plane in the memory wherein the plane is oriented between an orthogonal plane relative to the view of the object and a second plane which minimizes the distance to a three dimensional projection of the points in the sample image.

The predetermined orientation of the plane relative to the first and second planes is set by the user and may be half-way between the orthogonal first plane and the second plane. This allows the selected projection plane to be optimised for the relevant object.

The processing means may be further configured to generate shadowing for each of the objects by mapping a shadow texture over the sample image mapped to the polygon by performing multi-texturing. This advantageously provides shadowing with minimum additional processing cost.

The proportion of shadow texture covering the sample image may be computed by comparing a first value representing the height of the object with a second value representing the height of shadow at the corresponding location in the virtual environment. This provides a convenient way of calculating shadowing for the agents.

The processing means may be further configured to generate shadowing for each of the objects by: selecting a sample image from the memory depending on the angle of a virtual light source to the object; computing a second polygon representing the location of the shadow in the environment; and mapping a darkened sample image to the second polygon. This provides a method of providing shadows cast by the agents without the need to generated shadow texture, by re-using the sample images already stored.

For a better understanding of the present invention, specific embodiments will now be described, by way of example, with reference to the accompanying drawings, in which:

Fig. 1 shows a rendering system of the prior art;

Fig. 2 shows a hemispherical arrangement for sampling views of an object to obtain a set of sample images as shown in Fig. 3;

Fig. 3 shows a set of sample images of an object when viewed from a number of discrete different viewing angles;

Fig. 4 is a graph of the results of a prior art simulation using a small city model of 2368 polygons, showing how the

rendering time per frame increases with the number of dynamic agents;

Fig. 5 is a graph of the results of a prior art simulation using 10,000 dynamic agents, showing how the rendering time per frame increases with the complexity of the city model (from 267 polygons to 32128 polygons);

Fig. 6 shows a rendering system according to an embodiment of the present invention;

Fig. 7 shows how the error in a projection plane of a point from a three-dimensional object varies as the viewing angle changes;

Fig. 8 shows a three-dimensional projection of a view of an object with (a) a plane orthogonal to the viewing direction, and (b) a plane which minimizes the distance to the three-dimensional projection of the visible points;

Fig. 9 shows a display of a virtual environment including two textured impostors (impostor polygons with mapped sample images), and inset a set of compressed sample images;

Fig. 10 shows (a) a sample image, (b) the sample image with alpha values, (c) reproduction four times of a sample image by varying the colour of the regions indicated by the alpha values;

Fig. 11 shows a display of a simulated virtual environment;

Fig. 12 shows (a) a shadow volume height map, and (b) a static-object height map;

Fig. 13 shows (a) a shadowing texture, (b) a sample image with a white shadowing texture applied, (c) a sample image with a dark shadowing texture applied, and (d) a sample image with the shadowing texture of (a) applied;

Fig. 14 shows (a) the shadow heights for adjacent cells, (b) interpolated shadow heights for adjacent cells, (c) cells for computing a weighted average shadow height; and

Fig. 15 shows (a) a display of a virtual human with a shadow, (b) the sample image used to generate the shadow in (a), (c) shows the sample image projected onto an impostor polygon to generate the shadow in (a).

Fig. 6 shows a rendering system 20 according to an embodiment of the invention for rendering a multiplicity of dynamic three-dimensional objects in a virtual environment, the system comprising a memory 15 and a processor 16

The processor controls the rendering system 20 to perform operations in two distinct phases, a preprocessing phase 21 and a run time phase 22. The preprocessing phase 21, is used to build and store sampled images in an image database 7 in the . In addition, best impostor planes are calculated and stored in a best impostor plane database 6.

In order to build the image database 7, non-uniform discretization (step 1) is carried out in which a pre-computed set of discrete angles for viewing an object are used to determine the angles for image sampling. Then, the number and placement of the viewing angles are modified

slightly (ad-hoc discretization step 2) to adapt the viewing angles better to the shape and topological characteristics of the particular type of object, using an object-specific set of viewing angles. The object-specific viewing angles in the embodiment are input by the user, but may instead be computed using any appropriate method. These two steps together determine the particular angles at which image sampling (step 3) will be carried out. The sample images are pre-rendered ray traced images each of 256x256 pixels of the object (such as a human character) obtained using 3dmax with an orthographic projection. The sample images represent different views around the object from various positions and elevations. The sample images are then compressed (step 4) before storing in the dedicated texture memory in the image database 7.

One of the major drawbacks of the approach previously described by Tecchia [1] was the excessive amount of texture memory required to store the sampled images. All the sampled images were the same size even though in some of the images, the object (i.e. the human character) fitted into a much smaller area than in others. The sample images were stored in the texture memory using identical sized areas on a regular grid, which results in a considerable waste of space as can be seen for the texture shown in Figure 3. The advantage of the arrangement was that extracting a particular sample image was a trivial and fast operation. However, the disadvantage is that there is a lot of unused space surrounding some of the sample images.

In the embodiment, the original sampled images are modified and reorganised during the image compression step 4, to reduce the amount of texture memory required. The smallest

rectangle containing the relevant object portion (i.e. the human character) is computed for each sample image, and excess background pixels outside this are discarded. The revised images are then reorganised into a single image, to minimize the unused space, as shown in the inset 23 of the image shown in Fig. 9. The resulting image 23 is much smaller than the corresponding set of original images, with no loss in terms of visual quality. With this strategy, it is possible to reduce the amount of texture memory required to store the sample images to 25% of the original value. In order to achieve a good trade-off between the image quality and the memory required to store the sample images, each frame of animation was stored using 512 x 512 pixels.

In the embodiment, storage space required was further reduced by storing the images in the OpenGL compressed format (GL_COMPRESSED_RGBA_S3TC_DXT3) [2] which gives a further memory compression ratio of 1:4. This ratio is extremely efficient, although the image does lose part of the quality. This compression format allows the alpha values to be retained, and encodes them using 4 bits. When the final image is loaded into the texture memory in the image database 7, each frame of animation for one object type requires 256 Kbytes.

Since there is no longer a regular grid structure for storing the sample images, appropriate co-ordinates are also computed during the preprocessing stage and stored for use when retrieving each sample image. Also, during run time it is necessary to compute the correct size and orientation for each impostor in order to avoid the introduction of distortions of the sample image. It is noted that because of the optimal image storage strategy, the size and orientation

for the impostor polygon may vary for different frames of animation even though the viewing angle of the object remains fixed.

When image-based representations are used to render complex objects, two forms of undesirable artefacts can arise: missing data due to inter-occlusion may cause black regions to appear, and "popping" effects may occur. In order to maximize the rendering speed by reducing the geometrical complexity of the display, a single impostor polygon is chosen as the plane onto which a sample image is projected. As a result of this, the main perceived artefact is "popping" between different sample images as the viewpoint changes. Possible methods used for reducing the "popping" are blending together different sample images, or increasing the number of samples. However, it is undesirable to increase the number of sample images as this would increase the texture memory requirements. The use of multi-pass rendering (described later) prevents the images from being blended together.

"Popping" occurs because only a discrete number of sample images are taken. All the points on the surface of an object have been projected onto the same plane, in the particular view direction used when the sample image was created. As the camera position in the simulation changes, the projected points displayed on an impostor do not change, and therefore the impostor will no longer be a true 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, as shown in Figure 7. When the user's attention is focused on a single dynamic agent in a simulation it is possible to perceive a popping effect between frames.

The plane used in Tecchia [1] for an impostor polygon, onto which a sample image was projected, was the plane perpendicular to the direction of view from which the image was taken. However, this does not take into account the shape of the object or any kind of special occlusion that could be present in the image. In contrast, the embodiment uses a different approach in which (step 5) a best impostor projection plane with respect to each sample image is identified and stored in the best impostor plane database 6.

Firstly as shown in Figure 8, all the visible points in a sample image (marked "o") have been projected back into 3D space to obtain a three-dimensional form of the visible parts of the object. (Fig 8(a) shows a plane perpendicular to the view direction from which the image was taken.) In the embodiment, a projection plane is searched for which minimizes the sum of the distances from the sample points to the plane, known as principle component analysis (PCA). This identifies the two principle eigenvectors which define the best fitting plane, which is shown in Fig. 8(b).

In the case of sample images for polygonal models of human characters, using this best fitting projection plane results in a far better approximation of the position of the visible pixels in respect to the actual point positions in 3D. Unfortunately, other visual artefacts can occur when this best fitting plane is used as the impostor plane. For example, this new plane can cause an asymmetric warping of the image depending on which direction the camera moves during run time away from the sampling position. For some extreme cases, for a particular plane orientation and a certain distance of the camera, the perspective distortions

can also become too evident. Accordingly, the best fitting plane computed using PCA is itself not used, but is instead combined with the plane perpendicular to the view direction from which the sample image was taken. The resulting combined plane is known as the best impostor plane, and is located half-way between the best fitting plane and the orthogonal plane for the sample image. This best impostor plane is stored in the Best Impostor Plane Database 6 for retrieval later. Accordingly, the use of this plane minimises the popping whilst also limiting the introduction of other undesirable artefacts.

Although in the embodiment, the final orientation of the best impostor plane is chosen as half way between the two planes, this can be changed by the user to any other amount as desired depending on the type of object viewed (human character, animal, etc).

During run time, the current camera position and orientation of an agent are used to select the closest viewing angle of an agent from a precomputed set of discrete viewing angles (non-uniform discretization 8). The viewing angle is then modified slightly (ad-hoc discretization step 9) to adapt the viewing angles better to the topology of the particular type of object, using an object-specific set of viewing angles. The modified viewing angle is used to select the appropriate image from the Image Database 7, and also to select the corresponding correct best impostor plane from the Best Impostor Plane Database 6. The best impostor plane is placed in 3D space.

At step 12, multi pass colour filtering is carried out in which different regions of the same impostor are rendered

multiple times to define the colour for different parts of the models. In order to identify the areas to change, an alpha channel image was computed during preprocessing and stored as part of the sample image. A sample image (of a human character) rendered with ray tracing using 3dmax is shown in Fig. 10(a). The image in Fig. 10(b) is the human character with alpha-channels to identify the parts to modify. Fig. 10(c) shows the resulting images when the multi-pass rendering with respect to the alpha-channel is carried out. The final result is four human characters which all look different even though they have been built from the same 3D model.

Tuning the alpha channel, allows up to 256 different regions to be defined if no compression is used, or up to 32 different regions if the S3TC compression is used since only four bits are available for the alpha channel. During the rendering process for a virtual human, the alpha channel is used to select the parts to be rendered while multi-pass rendering is carried out. For each pass, the impostor polygon colour is changed and the texture (from the sample image) is applied using a flag GL_MODULATE and setting the alpha threshold of the alpha test to the value associated with the part of interest. Because the texture is modulated, the shading is preserved since it is already included as part of the sample image. Once the multi-pass rendering for one virtual human was complete, the next virtual human was rendered.

In the embodiment, multi-pass rendering was carried out up to three times resulting in changing only the colour of the shirt and the trousers. If required, more passes could be done since up to 32 different regions can be identified.

However, as this multi-pass rendering may slow down the overall rendering rate, a trade-off between the required variety of the agents and the rendering time must be reached.

Multi-pass rendering was used to draw, for example, the skin, trousers and shirt on different passes. Each agent was completed before moving on to the next one. The colours used will depend on the colours stored for that particular agent, in this way every agent can be different, even when there are 10,000 of them.

Finally, after a the dynamic agent has been rendered, shadow computation step 13 is carried out with respect to one or more light sources.

When the rendering is complete, the simulation may be viewed on the display 14.

The embodiment was implemented on a personal computer (Pentium III - 800 Mhz) with a NVIDIA GeForce GTS2 video card. The results are shown in table 1 below. The virtual environment was populated with six different dynamic agents, using three different passes at rendering to draw different (randomly chosen) colours. The simulation was run using an identical path to move through the virtual environment each time, and included a collision detection algorithm (not discussed here) performed for each of dynamic agents. An example of the display of the simulation is shown in Fig. 11.

No of dynamic agents in the simulation.	0	1,000	5,000	10,000
Frames per second.	26.20	24.08	18.26	13.35

Table 1.

The static environment was modelled using 41,260 polygons, the display being updated between 12 and 20 frames per second depending upon the displayed polygonal complexity. Simulations using 1000, 5000 and 10,000 dynamic agents were run. From the table, it is evident that the rendering of the environment itself uses much of the processing resources since the display ran at about 26 frames per second when there were no dynamic agents.

In the embodiment, the rendering speed is independent of the complexity of the models for the agents. In contrast, in the prior art, rendering the same number of agents is impossible using polygonal methods. Improvements over the prior methods have been obtained by the following aspects of the embodiments: firstly, the choice of the impostor is adapted to the object to render, thus minimising popping effects when the view is changed. Secondly, the amount of texture memory required per object has been reduced, thereby allowing a much larger number of different agents to be displayed. Finally, the embodiment uses a multi-pass rendering system, taking advance of the alpha channel to select and colour different regions of the agents, thereby also providing visual variety (the 10,000 virtual humans in the simulation are all different).

Multi-past rendering could also be used in the simulation of simple animation such as a virtual human turning their head from left to right. Here the virtual humans would be represented by two different impostors, one for the head and one for the body. The fact that the two are related would make various optimisations possible, and they would not be

considered as two completely separate objects. The two impostor polygons would probably be coplanar (although not necessarily), and might slightly overlap. Impostor A (for the head) could contain two alpha regions: the hair and skin. Impostor B (for the body) could contain three alpha regions: the skin, shirt and trousers. Multi-pass rendering is carried out as follows:

Impostor A, pass 1 - render the hair
Impostor A, pass 2 - render the skin
Impostor B, pass 1 - render the skin
Impostor B, pass 2 - render the shirt
Impostor B, pass 3 - render the trousers

Although the specific embodiments relate to the real-time display of crowds in a virtual city, the techniques described are equally applicable for any virtual environment in which a large non-static population is displayed. However, it should also be noted that when the number of dynamic agents is reduced for example, if the new point is zoomed in closer and some individuals become more visually important, then higher detail solutions [36,37] should be used instead or in combination with these techniques.

Finally, at step 13, the system 20 performs computation for the shadows and shadowing of the dynamic agents. Although, in the context of a complex virtual environment, four different cases or shadow computation can be distinguished, see below, the description below deals only with the shadow computation in case (b) and partly case (c):

(a) Shadows from the static onto other static geometry, such as a model of a building casting a shadow onto the ground;

(b) Shadows from the static onto the dynamic geometry, such as the shadow from a model of the building falling onto the dynamic (virtual humans);

(c) Shadows from the dynamic onto the static geometry, such as the shadows the dynamic agents (virtual humans) cast onto the ground; and

(d) Shadows from one dynamic object onto another dynamic object, such as the shadow from one virtual human falling onto another virtual human.

In the embodiment, case (a) was addressed using fake shadows [15] to display shadows from static objects (e.g. buildings) onto the ground and the OpenGL lighting to shade the static objects. However, these methods do not allow shadows to be computed when the light sources are moving.

Case (d) is not addressed in this application. It is particularly difficult and processing-time intensive to compute shadows between thousands of dynamic agents.

Consider firstly, the shadow computation for case (b), in which the dynamic agents move in and out of shadows cast by static objects (buildings). This problem is extremely complex, since there may be thousands of dynamic agents which need to be updated every frame. In the embodiment, a 2.5D model of the city is used, essentially a two dimensional grid with each cell containing a value representing the height of the item occupying that cell. When the dynamic agents move through the virtual environment, the height information of each target cell is checked for obstacles, and allows for fast collision detection against both static objects and other dynamic agents [3]. In order to perform shadow computation, a 2.5D grid is also used to store shadow

information. As a dynamic agent moves through the virtual environment, the shadow height information stored for the current cell it occupies is compared with the height of the agent itself in order to determine whether the agent is completely or partially shaded, or not shaded at all.

Firstly, in order to obtain the 2.5D map representing shadows, shadow planes which define the volumes of shadows are computed for a given light source, using one of a number of standard ways. In the embodiment, shadow volumes were computed by firstly defining a plane located below the virtual environment and then computing the intersections with this plane of every ray from the light source to every vertex in the virtual environment.

After the shadow planes have been computed, rendering is performed from a viewpoint located directly above the virtual environment, using an orthogonal projection. From the z-buffer of this image, height information for the shadows is extracted and stored on a normal resolution 2D grid, forming a shadow volume height map as shown in Figure 12 (a). The corresponding static object height map is shown in Figure 12 (b). The height of the shadows above objects is therefore given by the difference between the shadow volume height map and the height of the static object located at a corresponding cell of the 2D grid. The resulting difference map is know as a shadow height map, and will be zero in places where the static environments is not in shadow. This is the map used in the following methods.

From the 2D shadow height map, it is possible to compare the height of dynamic agents moving through the environment with the height of the shadow at that position and to compute the

proportion of the agent which is in shadow. A dynamic agent moving through the virtual environment is located on a particular cell of the 2D grid, although they can also occupy one of a discreet set of positions within a single cell to allow for smoother animations.

For each cell occupied by a dynamic agent, the height of the agent is compared with the height of the shadow as stored on the 2.5D shadow height map. This gives us the percentage of the agent in shadow. It is then necessary to compute the shading of the agents. In traditional techniques, using polygonal models a straightforward way to shade the agent is to go back to the polygonal model and regenerate the images. Alternatively, another method is to pre-compute different lighting configurations and to save images for each of them.

In contrast, in the embodiment shadow texture is mapped onto the impostor for the dynamic agent to darken the part in shadow. A sample texture for use when the agent is partially shaded is shown in Figure 13(a). In this case, a single texture representing two colours, white on top and black underneath, is used.

Alternatively, if the agent is fully shaded or not shaded at all, the impostor polygon can be rendered in either white or grey. In the solution of the embodiment, the assumption is made that when a shadow is cast, everything underneath is covered, due to the 2.5D configuration of the scene. This excludes the cases of roofs overhanging edges of buildings, bridges, and any other kind of "non 2.5D" objects.

As explained, a texture using only two colours as shown in Figure 13 (a) is used. This texture is mapped to the

impostor for the agent using multi-texturing features of the hardware (graphic cards). Since this multi-texturing is virtually processing-cost free, a second texture is rendered for every impostor, regardless of whether the impostor is fully or partially shaded or not shaded at all. Before the texture is mapped to the impostor, it is necessary to compute the relative proportions of light and dark (described in more detail later). In this way it is possible to use a white texture for impostors which are not shaded at all, and black texture for completely shaded impostors. The resulting impostors are shown for the unshaded (b), completely shaded (c) and partially shaded (d) cases in Figure 13.

However, the shadow texture in Fig. 13(a) shows a horizontal split between the light and dark regions, but shadows rarely describe horizontal boundaries in virtual environments lit by an elevated source (for example, a city scene lit by the sun). To detect the inclination of the shadow, cells surrounding the occupied cell are checked for differences in height of the shadow, shown in Figure 14. Interpolating between adjacent cells (b) can provide smoother boundaries of the shadowing to avoid shadow "popping" between adjacent cells. However, this does also result in some inaccuracies in the shadow boundary. Interpolation is performed by calculating an average of the heights of the shadows of the surrounding cells, weighted by the position of the agent within the current cell (shown as the circular dot in the upper left quadrant of the central cell in Figure 14 (c)). A shadow boundary is computed for each side of the impostor polygon, and this is done at rendering time because the borders of the impostor are view dependent.

The faster computation, the area comprising the current cell and eight adjacent cells is divided into four quadrants, as shown in Figure 14 (c). The weighted average is then calculated using only the cells containing part of the quadrant in which the agent is located. This interpolation allows a better description of the inclination of the shadow, and also allows smoother transitions when the agent moves.

Secondly, shadow computation is used to compute shadows cast by the dynamic agents themselves onto the ground of the virtual environment. For these shadows, it is possible to use the same sample images which have already been computed for the agents. This is because, although the images were created by sampling views of the object from discrete viewing angles, each viewing angles can be considered equivalent to a point light source such that a silhouette of the view from that angle gives the shadow of the object when projected onto an appropriate plane. This method enables high quality shadows representing the exact shape of the dynamic agent to be computed with extremely low additional processing cost. The shadow is therefore generated by selecting the appropriate sample image, and mapping this onto a projected shadow plane (another impostor polygon).

In the embodiment, a flat polygon is chosen and located on the ground of the virtual environment, such that the bottom of the dynamic agent and the corresponding shadow always meet. This is illustrated in Figure 15. The final version of the shadow computed for the dynamic agent is shown in Figure 15 (a). To achieve this the correct sample image 25 is first selected based on the "viewing position" of the light source and the correct frame of animation. A shadow impostor polygon 26 is projected onto the ground and the

sample image texture 25 is set with a dark colour and mapped to the projected polygon. Using this method, no new textures are required to be generated.

Using the method described, the ground onto which the shadow is cast is assumed to be horizontal. This means that shadows incorrectly miss static objects and other dynamic objects. However, it is also possible to detect these changes using grid discretisation to check the coverage in cells adjacent to the projected polygon and therefore more accurate representations of shadows can be computed for the "important" objects closer to the viewer.

If a set of projected polygons are pre-computed for each possible light source position, then shadows can be displayed interactively for a moving light source. If it is desirable to have flying dynamic agents which do not touch the ground of the virtual environment, then the method can be extended by moving the projected polygon accordingly along the ground.

The embodiment was tested using a PC with a Pentium 833 Mhz processor and a NVIDIA GeForce GTS2 64 Mb graphic card. The virtual environment was modelled using 41,260 polygons and thousands of virtual humans, including collision avoidance with the building and each other. Tests were carried for 1000, 5000 and 10,000 virtual humans, and a display window set to 1024 x 1024 pixels.

Creating the 2.5D shadow map took 1 second for such a model, including the shadow volume vertices computation (0.1 second). The shadow map was 1024 x 1024 pixels, and the stored heights represented using 24 bits.

To evaluate the additional cost of rendering shadows, different components of shadow computation and display were activated step by step in order to highlight time consumption required for each different step of the method. A fixed walk through path was also set since the number of virtual humans and polygons renders during each frame impacts on the rendering frame rate. Each simulation was composed of 1160 frames, and in each case the average number of rendered frames per second was calculated as well as the total computation time for all 1160 frames. The results are summarised in Table 2.

No. of virtual humans	Display (with no shadows)		+ displaying fake shadows		+ displaying shadow of virtual human on ground		+ computing and displaying shadowing of virutal humans	
	Fps	ms	Fps	ms	Fps	ms	Fps	ms
0	26.20	44263	19.78	58635				
1,000	24.08	48159	18.87	61469	18.57	62460	18.32	63311
5,000	18.26	63521	15.29	75829	14.57	79745	12.45	93164
10,000	13.35	86845	11.67	99363	10.57	109718	8.48	136787

Table 2

The second column shows the results of the simulation without any shadows at all. The third column shows the results of the simulation displaying only fake shadows [15] (i.e. shadows cast by static objects onto the ground). The fourth column shows the results of the simulation which additionally included the shadows cast by the virtual humans on the ground. The fifth column shows the results of the simulation additionally including the shadows cast by static objects onto the virtual humans.

The display of the static model, as well as displaying the fake shadows appear to be the most time consuming task. The table shows that in all cases the frame rate decreases when more virtual humans are included in the simulation. The additional cost of the shadow computation and display indicated in columns four and five is fairly low, even when updating 10,000 dynamic virtual humans. One of the major advantages of the implementation described is the combination of the display of the impostor for the dynamic agent with the shadow computation. This means that the additional cost of the shadows is independent of the light position.

Another interesting implementation detail is that when displaying the dynamic agents including shadows, the relevant portion of the texture memory only needs to be loaded once. Since the shadow corresponds directly to the position of the dynamic agent, the cost of displaying the shadow for each virtual human is reduced to the display of one polygon. Since multi-texturing is used, displaying the shadowing onto the dynamic agent does not add to the cost even if the dynamic agent is not in shadow since it is the second texture of the multi-texturing.

The coordinates to map the shadowing texture onto the dynamic agents are computed at each rendering frame. The projected shadow polygon on the ground for displaying the shadows cast by the dynamic agents is pre-computed.

The method of shadow computation using a 2.5D map to locate the shadows can be used for any type of dynamic scenes with 2.5D configuration, and is not restricted to scenes in which the dynamic agents are simulated using impostors.

The embodiment currently only relates to the computation of sharp shadows using a single light source. However edges of the shadowing can be blurred to obtain soft shadows, and the method may also be extended to moving light sources.

The processes described above may be performed by a computer program running on a computer in the embodiment described. Such a computer program can be recorded on a recording medium (for example a magnetic disc or tape, an optical disc or an electronic memory device, such as a ROM) in a way well known to those skilled in the art. When the recording medium is read by a suitable reading device, such as a magnetic or optical disc drive, a signal is produced which causes a computer to perform the processes described.

The processes may also be performed by electronic means.

The following documents referred to in the description are hereby incorporated into the specification by reference:

[#1.] F. Techia and Y. Chrysanthou. Real-time Rendering of Densely Populated Urban Environments. In Péroche and Rushmeier, editors, Rendering Techniques '00 *(10th Eurographics Workshop on Rendering)*, pages 45-46, Brno, Czech Republic, June 200, Springer Verlag.

[#2.] OpenGL texture compression. http://os.sgi.com/projects/olgsample/registry/EXT/texture_compression_s3tc.txt.

[#3.] [Techia UK 2000] F. Techia and Y. Chrysanthou, Real-time Visulisation of Densely Populated Urban Environments: a

Simple and Fast Algorithm for Collision Detection, *Eurographics UK*, Swansea, April 2000.

[#4.] [Slater 1995] M. Slater, M. Usoh, Y. Chrysanthou (1995) The Influence of Shadows on Presence in Immersive Virtual Environments, Virtual Environments '95, M. Goebel (ed.) Springer Computer Science, ISSN 0946-2767, pp8-21.

[#5.] [Kallman 2000] M. Kallmann, J-S. Monzani, A. Caicedo, D. Thalman, ACE: A Platform for the Real Time Simulation of Virtual Human Agents, EGCAS'200 - *11th Eurographics Workshop on Animation and Simulation*, Interlaken, Switzerland, August 2000.

[#6.] [Farenc 1999] N. Farenc, R. Boulic, D. Thalmann, An Informed Environment Dedicated to the Simulation of Virtual Humans in Urban Context, Proc. ,*Eurographics '99*,, Milano, Italy, 1999, pp.309-318.

[#7.] [Friedman 1995] S. Friedman W. Jepson, R. Ligget. An environment for real-time urban simulation In *Symposium on Interactive 3D Graphics*, ACM, Monterey CA USA, 1995.

[#8.] [Donikan 1994] S. Donikan and B Arnaldi, Complexity and concurrency for behavioral animation and simulation. In G. Hgron and O. Fahlander, editors, *Fifth Eurographics Workshop on Animation and Simulation*, Oslo Norv?ge, September 1994.

[#9.] [Donikan and Rutten 1995] S. Donikan and E. Rutten. Reactivity, concurrency, data-flow and hierarchical preemption for behavioural animation. In E.H. Blake R.C.

Veltkamp, editor, *Programming Paradigms in Graphics '95*, Springer-Verlag, *1995*.

[#10.] [Donikan and Curzot 1995] S. Donikon and R. Cozot. General animation and simulation platform. In D. Terzopoulos and D. Thalmann, editors, *Computer Animation and Simulation '95*, pages 197-209, Springer-Verlag, 1995.

[#11.] [Lischinski 1992] Lischinski, D., Tampieri, F., and Greenberg, D. P. (1992). Discontinuity meshing for accurate radiosity. IEEE Computer Graphics and Applications, 12(6):25{39}.

[#12.] [Drettakis 1994] Drettakis, G and Fiume E. (1994). A fast shadow algorithm for area light sources using backprojection. In Glassner, A., editor, ACM Computer Graphics, pages 223-230.

[#13.] [Soler 1998] Cyril Soler, Françoise Sillion, Fast Calculation of Soft Shadow Textures Using Convolution, Computer Graphics Proceedings pages 321--332, Jul 1998.

[#14.] [Heckbert 1997] Paul Heckbert, Michael Herf. Simulating Soft Shadows with Graphics Hardware. CMU-CS-97-104, CS Dept, Carnegie Mellon U., Jan 1997.

[#15.] [Blinn 1998] Blinn, J.F. (1988). Jim Blinn's Coner, Me and my (fake) shadow. IEEE Computer Graphics and Applications, 8(1):82-86.

[#16.] [Chin 1989] Chin, N and Feiner, S. (1989). near real-time shadow generation using BSP trees. ACM Computer Graphics, 23(3):99-106.

[#17.] [Chrysanthou 1996] Chrysanthou, Y. (1996). Shadow Computation for 3D Interaction and Animation. PhD thesis, Queen Mary and West-eld College, University of London.

[#18.] [Crow 1977] Crow, F. (1997). Shadow algorithms for computer graphics. ACM Computer Graphics, 11(2):242-247.

[#19.] "Micheal D. McCool", "Shadow volume reconstruction from depth maps", pages 1--26, "ACM Transactions on Graphics", volume = 19, number = 1, year = 2000.

[#20.] Tim Heidmann, "Real Shadows, Real Time", Iris Universe, note = {Silicon Graphics, Inc., volume = {18}, year = {1991}, pages = {28-31}.

[#21.] [Williams 1978] L. Williams, Casting Curved Shadows on Curved Surfaces, 1978, August, Volume 12, Number 3, computer Graphics, pages 270-274.

[#22.] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario. Conservative visibility and strong occlusion for viewspace partioning of densely occluded scenes. *Computer Graphics Forum*, 17(3):243-254, 1998. ISSN 1067-7055.

[#23.] 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.

[#24.] François Silion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization

of urban scenery. *Computer Graphics Forum*, 16(3):207-218, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055.

[#25.] Germot Schaufler and Wolfgang Sturzlinger. A three-dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):C227-C235, C471-C472, September 1996.

[#26.] Johnathon Shade, Dani Lischinski, David Salesin, Tong DeRose, and John Snyder. Hierarchical image caching for acceleted 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, Lousiana, 04-09 August 1996.

[#27.] Shenchang Eric Chen and and Lance Williams. View interpolation for image synthesis. In James T. Kajiya, editor, *Computer Graphics (SIGRAPH '93 Proceedings)*, volume 27, pages 279-288, August 1993.

[#28.] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malic. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. in *ACM SIGGRAPH 96 Conference Proceedings*, pages 11-20, August 1996, held in New Orleans, Louisiana, 04-09 August 1996.

[#29.] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39-46, November 1995.

[#30.] William R. Mark. Leonard McMillan, and Gary Bishop, Post-rendering 3D warping. In Michael Cohen and David

Zelter, editors, *1997 Syposium on Interactive 3D Graphics*, pages 7-16, ACM SIGGRAPH, April 1997, ISBN 0-89791-884-3.

[#31.] Lucia Darsa, Bruno Costa Silva, and Amitabh Varshney, Navigating static environments using image-space simplification and morphing. In Michael Cohen and David Zelter, editors *1997 Symposium on Interactive 3D Graphics*, pages 25-34 ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

[#32.] G. Schaufler. Per-object image warping with layered impostors. In *9th Eurographics Workshop on Rendering '98*, pages 145-146, Vienna, Austria, April 1998. EUROGRAPHICS ISBN 0-8979-884-3.

[#33.] William J Dally, Leonard McMillan, Gary Bishop, and Henry Fuchs. The delta tree: An object-centered approach to image based rendering. Technical Memo AIM-1604, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, May 1996.

[#34.] A. Aubel, R. Boulic, and D. Thalmann, Animated imposters for real-time display of numerous virtual humans. In Jean-Claude Heudin, editor, *Proceedings of the 1st International Conference on Virtual Worlds (VW-98)*, volume 1434 of LNAI, pages 14-28, Berlin, July 1-3 1998. Springer.

[#35.] A. Aubel, R. Boulic, and D. Thalmann, Lowering the cost of virtual human rendering with structured animated impostors *In Proceedings of WSG 99*, Plzen, Czech Republic, 1999.

[#36.] D. Thalman, 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.

[#37.] Xiaoyuan Tu and Demetri Terzopoulos, Artificial fishes: Physics, locomotion, perception, behaviour. 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.

CLAIMS

1. A system for rendering a multiplicity of dynamic three-dimensional objects in a virtual environment, the system comprising a memory for storing a plurality of sample images representing the objects viewed from different angles, the system further comprising processing means configured for:

for each object to be displayed, selecting a sample image from the memory based on the angle of viewing the object, wherein the pixels of the sample image include an alpha channel value for identifying a plurality of different regions of the image;

mapping the sample image to a polygon representing the location of the object in the environment, including performing multi-pass rendering of the sample image, wherein each pass is for rendering differently coloured regions of the sample image identified by the alpha channel values.

2. A system according to claim 1, the processing means further configured to store in the memory in relation to each type of object at least one set of sample images representing one of the objects viewed from a set of discrete viewing directions.

3. A system according to claim 1 or 2, the processing means further configured to store in the memory in relation to each type of object a plurality of sets of sample images, the sets corresponding to different frames of animation of the object.

4. A system according to any of claims 1 to 3, wherein the processing means is further configured to perform image compression on an original set of sampled images before

storing the sample images in the memory, wherein the original sample images each comprise a background portion and an object portion, the compression comprises:

revising each of the sample images by discarding part of the background portion which is not enclosed by a smallest-fit rectangle around the object portion; and

reorganising the arrangement of storing the revised sample images to minimize unused storage space in the memory.

5. A system according to any preceding claim, wherein the processing means is further configured to:

select one of a plurality of projection planes stored in the memory based on the angle of viewing the object; and

map the selected sample image onto the selected projection plane.

6. A system according to any preceding claim, wherein the processing means is further configured to:

for each sample image, compute a plane for said image and store the plane in the memory wherein the plane is oriented between an orthogonal plane relative to the view of the object and a second plane which minimizes the distance to a three dimensional projection of the points in the sample image.

7. A system according to claim 6, wherein a predetermined orientation of the plane relative to the first and second planes is set by the user.

8. A system according to claim 6 or 7, wherein a predetermined orientation of the plane is half-way between the orthogonal first plane and the second plane.

9. A system according to any preceding claim, wherein the processing means is further configured to generate shadowing for each of the objects by mapping a shadow texture over the sample image mapped to the polygon by performing multi-texturing.

10. A system according to claim 9, wherein the proportion of shadow texture covering the sample image is computed by comparing a first value representing the height of the object with a second value representing the height of shadow at the corresponding location in the virtual environment.

11. A system according to any preceding claim, wherein the processing means is further configured to generate shadowing for each of the objects by:

selecting a sample image from the memory depending on the angle of a virtual light source to the object;

computing a second polygon representing the location of the shadow in the environment; and

mapping a darkened sample image to the second polygon.

12. A method for rendering on a computer system a multiplicity of dynamic three-dimensional objects in a virtual environment, the system comprising a memory for storing a plurality of sample images representing the objects viewed from different angles, the method comprising:

for each object to be displayed, selecting a sample image from the memory based on the angle of viewing the object, wherein the pixels of the sample image include an alpha channel value for identifying a plurality of different regions of the image;

mapping the sample image to a polygon representing the location of the object in the environment, including

performing multi-pass rendering of the sample image, wherein each pass is for rendering differently coloured regions of the sample image identified by the alpha channel values.

13. A method according to claim 12, further comprising storing in the memory in relation to each type of object at least one set of sample images representing one of the objects viewed from a set of discrete viewing directions.

14. A method according to claim 12 or 13, further comprising storing in the memory in relation to each type of object a plurality of sets of sample images, the sets corresponding to different frames of animation of the object.

15. A system according to any of claims 12 to 14, further comprising performing image compression on an original set of sampled images before storing the sample images in the memory, wherein the original sample images each comprise a background portion and an object portion, the compression comprises:

revising each of the sample images by discarding part of the background portion which is not enclosed by a smallest-fit rectangle around the object portion; and

reorganising the arrangement of storing the revised sample images to minimize unused storage space in the memory.

16. A method according to any of claims 12 to 15, further comrising:

selecting one of a plurality of projection planes stored in the memory based on the angle of viewing the object; and

mapping the selected sample image onto the selected projection plane.

17. A method according to any of claims 12 to 16, further comprising:

for each sample image, computing a plane for said image and storing the plane in the memory wherein the plane is oriented between an orthogonal plane relative to the view of the object and a second plane which minimizes the distance to a three dimensional projection of the points in the sample image.

18. A method according to claim 17, wherein a predetermined orientation of the plane relative to the first and second planes is set by the user.

19. A method according to claim 17 or 18, wherein a predetermined orientation of the plane is half-way between the orthogonal first plane and the second plane.

20. A method according to any of claims 12 to 19, further comprising generating shadowing for each of the objects by mapping a shadow texture over the sample image mapped to the polygon by performing multi-texturing.

21. A system according to claim 20, wherein the proportion of shadow texture covering the sample image is computed by comparing a first value representing the height of the object with a second value representing the height of shadow at the corresponding location in the virtual environment.

22. A method according to any of claims 12 to 21, further comprising generating shadowing for each of the objects by:

selecting a sample image from the memory depending on the angle of a virtual light source to the object;

computing a second polygon representing the location of the shadow in the environment; and

mapping a darkened sample image to the second polygon.

23. A computer program, comprising code instructions which when run on a computer system cause the computer to perform the method according to any of claims 12 to 22.

24. A rendering system, substantially as hereinbefore described, with reference to and/or substantially as illustrated in any one or any combination of the accompanying drawings.

25. A method for rendering a multiplicity of dynamic three-dimensional objects in a virtual environment, substantially as illustrated in any one or any combination of the accompanying drawings.

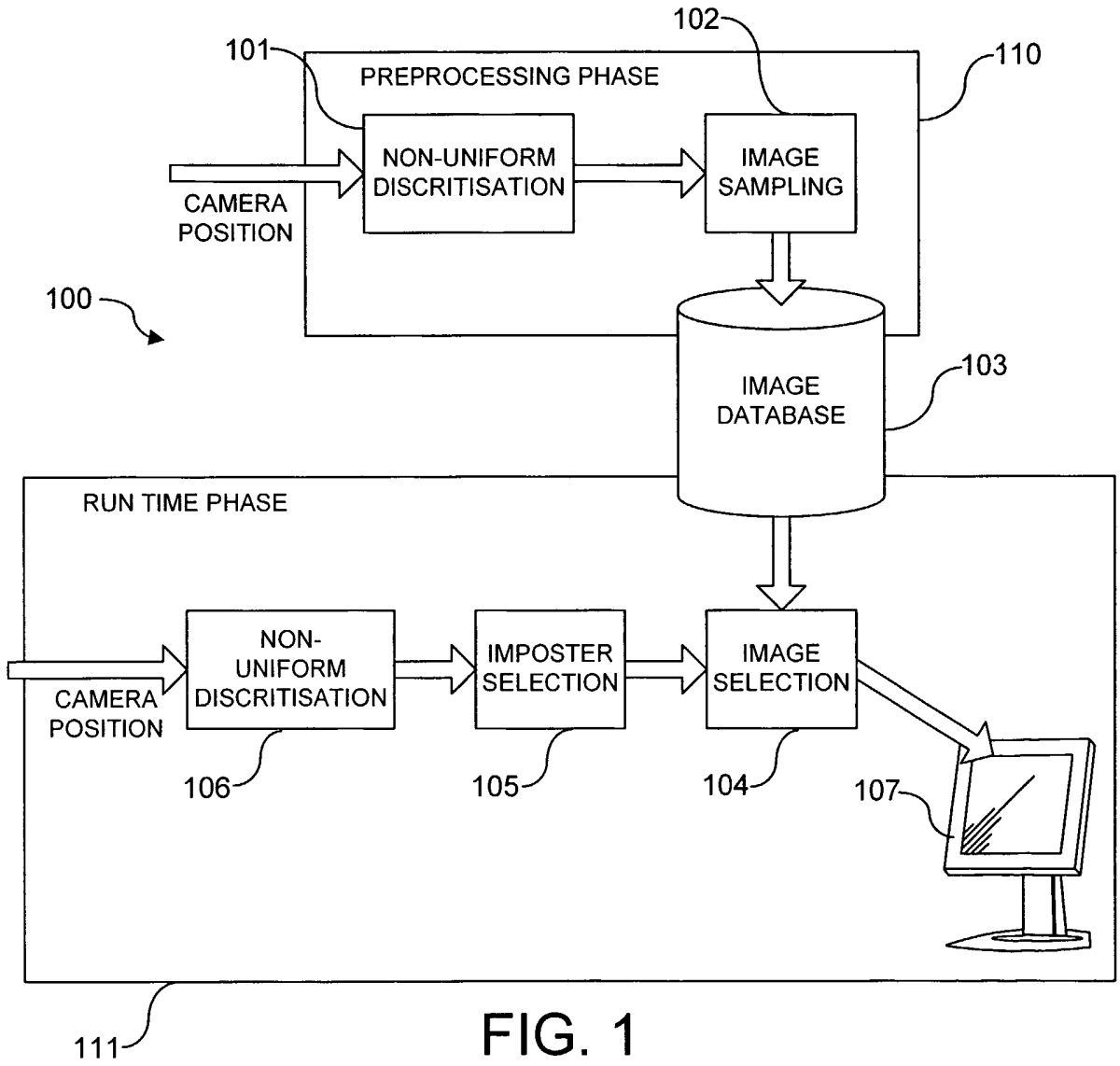

102
101
110
PREPROCESSING PHASE
NON-UNIFORM DISCRITISATION
IMAGE SAMPLING
CAMERA POSITION
100
IMAGE DATABASE
103
RUN TIME PHASE
NON-UNIFORM DISCRITISATION
IMPOSTER SELECTION
IMAGE SELECTION
CAMERA POSITION
106
105
104
107
111

FIG. 1

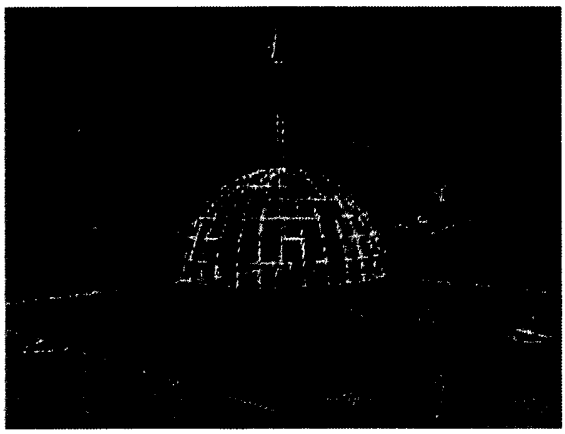


FIG. 2

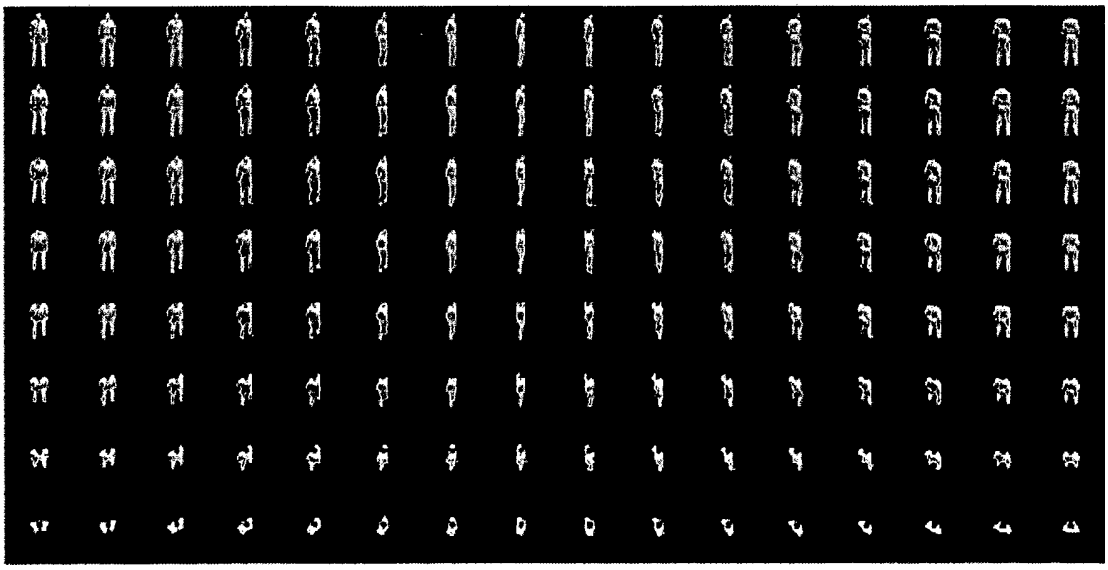


FIG. 3

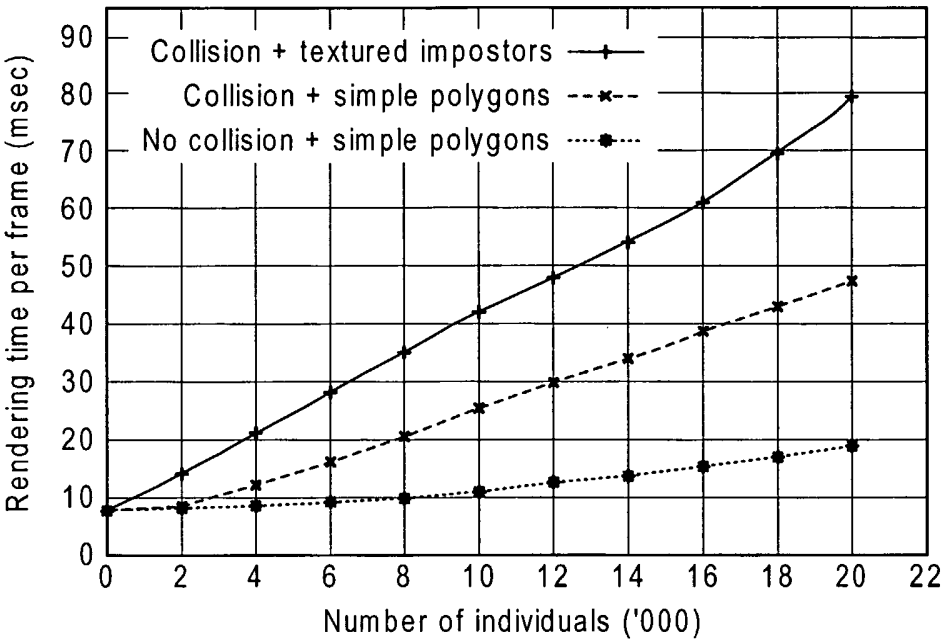

Collision + textured impostors
Collision + simple polygons
No collision + simple polygons
Rendering time per frame (msec)
0
10
20
30
40
50
60
70
80
90
0
2
4
6
8
10
12
14
16
18
20
22
Number of individuals ('000)

FIG. 4

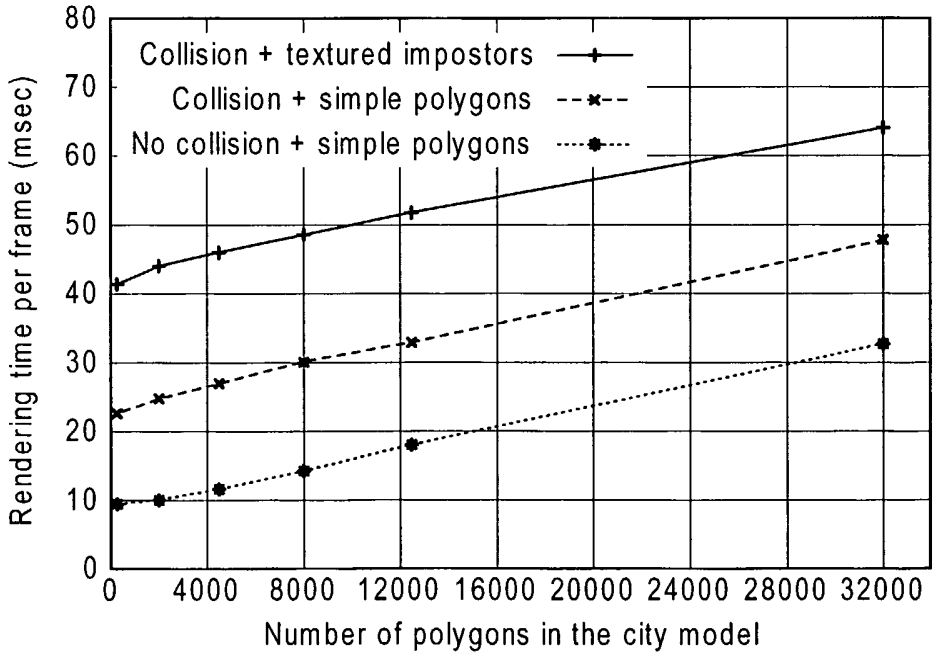

Collision + textured impostors
Collision + simple polygons
No collision + simple polygons
Rendering time per frame (msec)
0
10
20
30
40
50
60
70
80
0
4000
8000
12000
16000
20000
24000
28000
32000
Number of polygons in the city model

FIG. 5

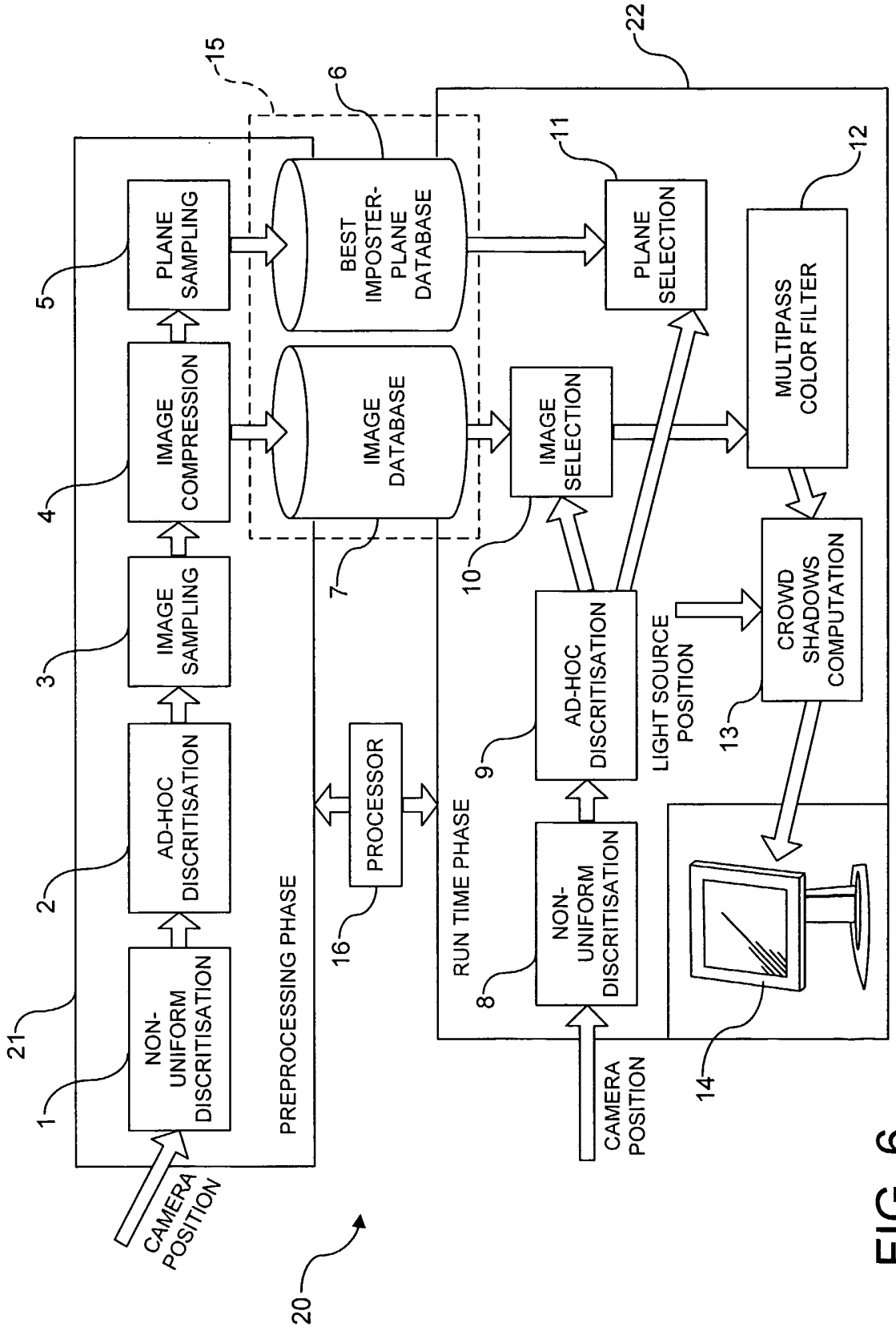

NON-UNIFORM DISCRITISATION
AD-HOC DISCRITISATION
IMAGE SAMPLING
IMAGE COMPRESSION
PLANE SAMPLING
BEST IMPOSTER-PLANE DATABASE
IMAGE DATABASE
PROCESSOR
PREPROCESSING PHASE
CAMERA POSITION
RUN TIME PHASE
NON-UNIFORM DISCRITISATION
AD-HOC DISCRITISATION
IMAGE SELECTION
PLANE SELECTION
LIGHT SOURCE POSITION
CROWD SHADOWS COMPUTATION
MULTIPASS COLOR FILTER
CAMERA POSITION
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
20
21
22

FIG. 6

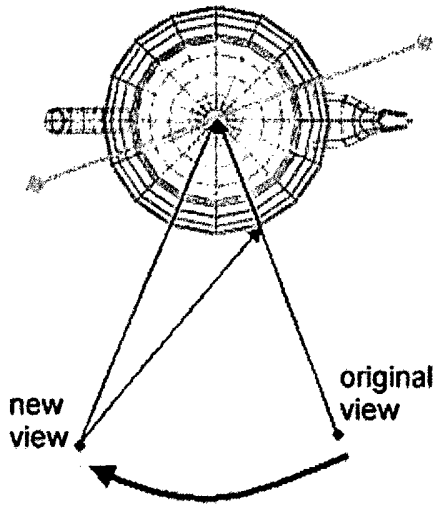

new
view
original
view

FIG. 7

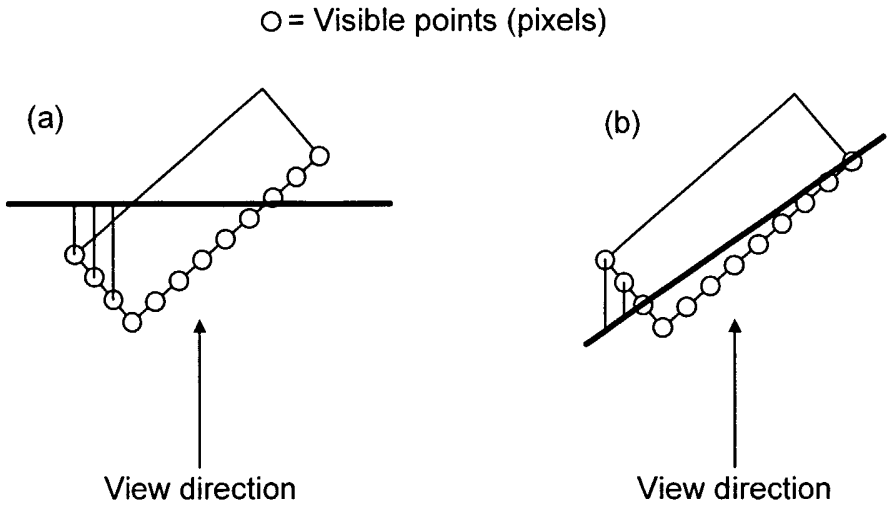

O = Visible points (pixels)
(a)
(b)
View direction
View direction

FIG. 8

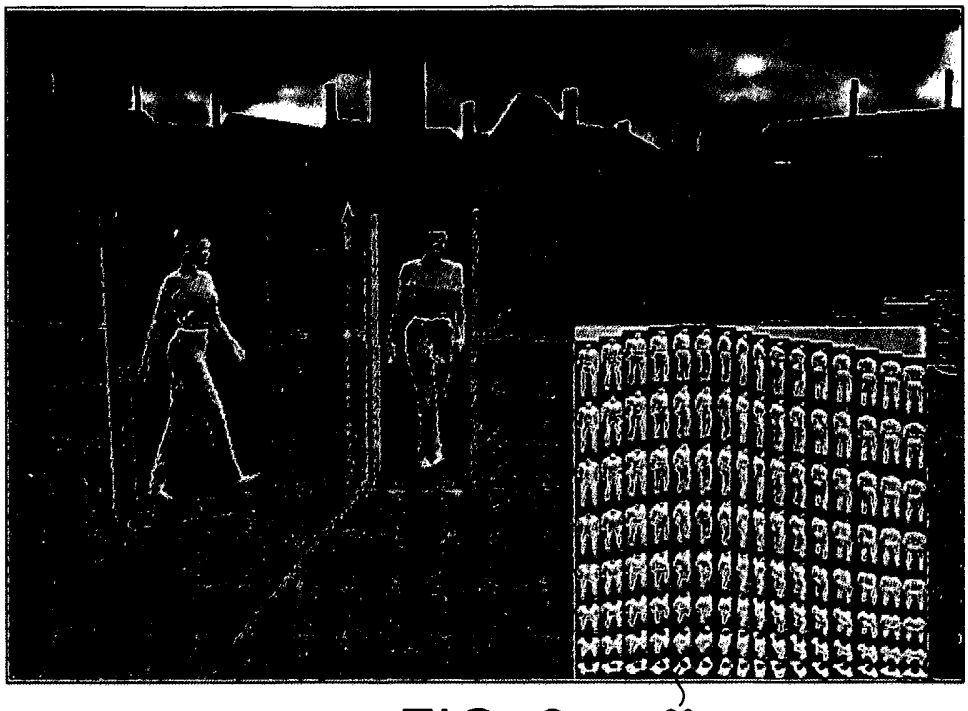


FIG. 9

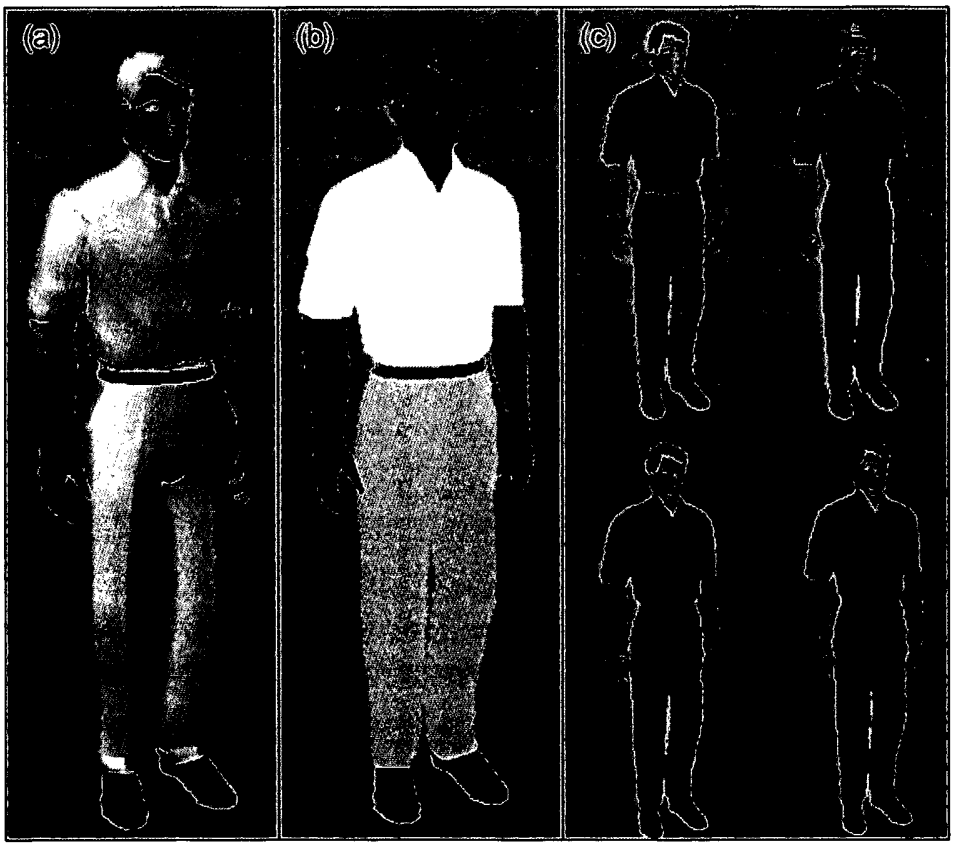


FIG. 10

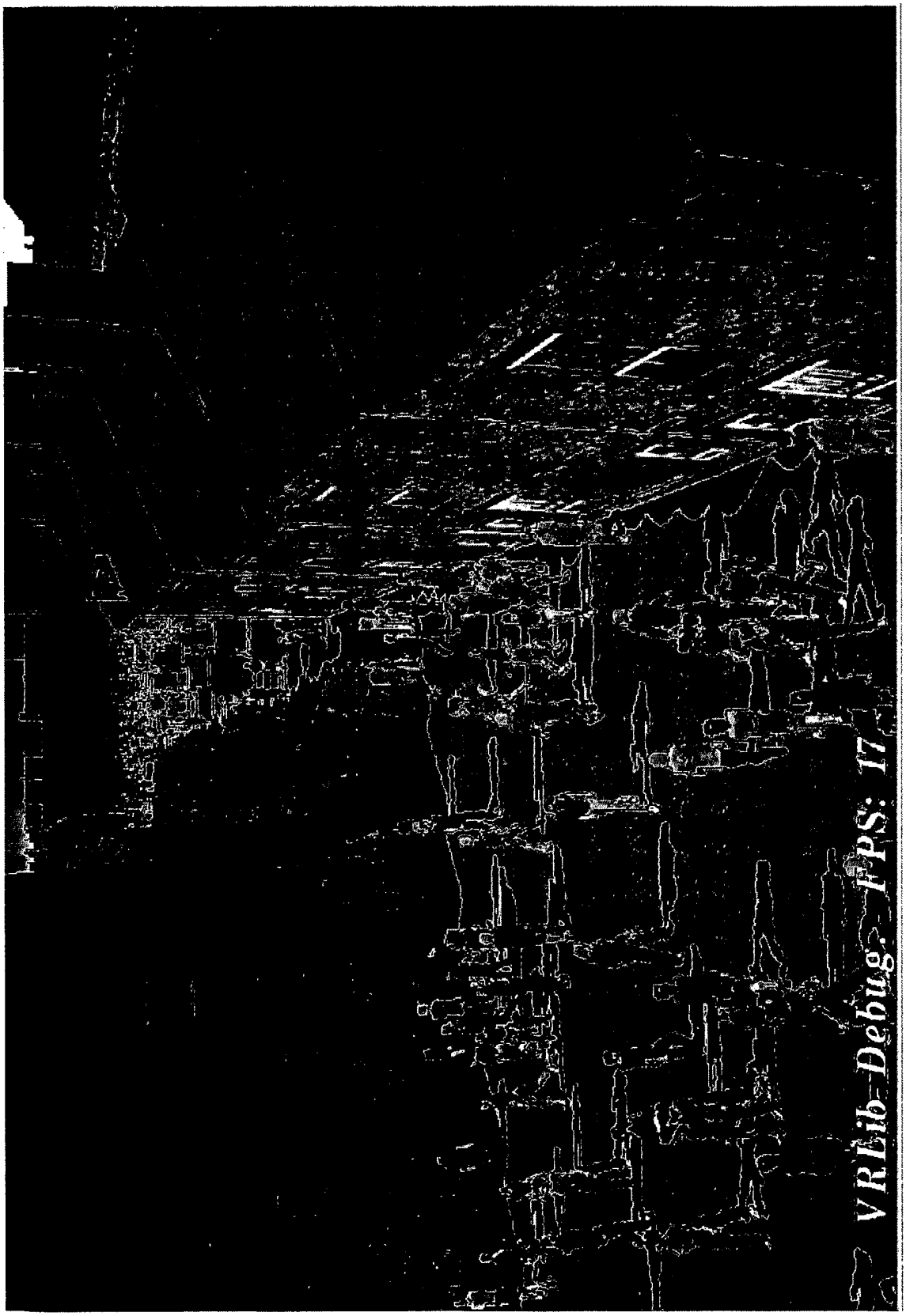
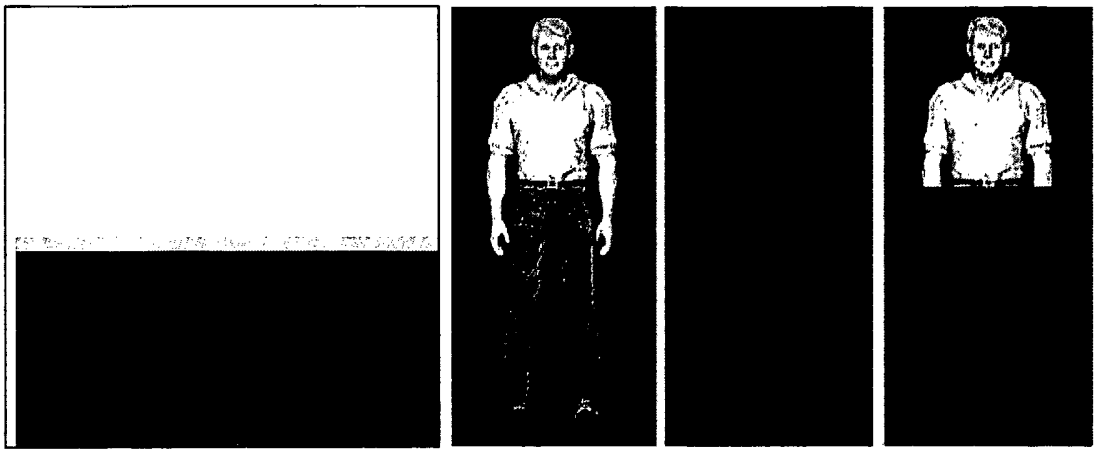


FIG. 11

(a) (b)

FIG. 12



(a) (b) (c) (d)

FIG. 13

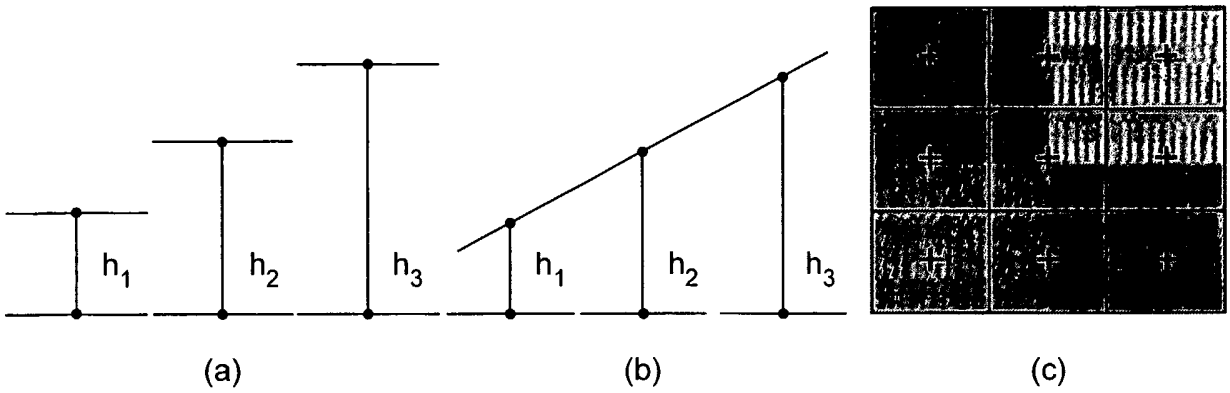


FIG. 14

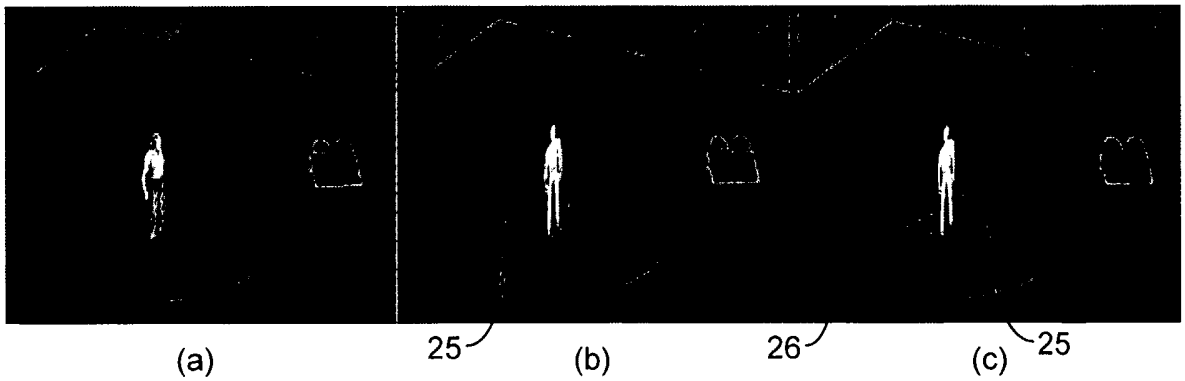


FIG. 15