Impostors-Based Real-Time Avatar Behavior in Virtual Reality Systems[*]

Yonggao Yang and Jim X. Chen
Department of Computer Science, George Mason University
{yyang2, jchen}@gmu.edu

Abstract

Real-time human-like avatar behavior in virtual environments makes important contributions to the realism of VR worlds. The existing human-like animation approaches demand intensive calculations and graphics rendering. Complex 3D avatar models are usually constructed with thousands of triangles, which slow down the rendering rate. In this paper, we present an impostors-based approach to achieve the real-time human-like avatar behavior in virtual environments. Theoretically, our method animates any human-like behavior in virtual worlds without creating a 3D avatar model. Experiments show that this approach consumes less time in both calculation and rendering, and thus leaves more CPU time and hardware resources to other tasks. It brings a substantial improvement to avatar behavior in VR worlds.

1. Introduction

Real-time human-like avatar behavior in virtual environments improves the realism of virtual reality (VR) worlds. Avatars that can do rich human-like activities and movements in real-time make virtual environments more interesting. The common required avatar behaviors in VR worlds are walking, running, jumping, and dancing. For example, navigating in a VR world demands the player's avatar walking. Intelligent avatars with behaviors controlled by computer may ask for randomly generated behavior. The two goals in avatar behavior in VR worlds are both "real-time" and "human-like". Obviously there is a contradiction between the speed and the realism of avatar behavior. It is hard to find a compromise between these two factors to achieve both real-time and human-like:

- **Real-time is mandatory.** Because of the very limited amount of available computer resources, usually we have to sacrifice realism for speed. VR worlds lay more emphasis on speed and interaction. The "real-time" here is a "soft" real-time, where system performance is degraded if the time constraint is not always met.
- **Human-like crowd avatar rendering required.** In a distributed virtual environment, multiple players appear in the same VR scene at the same time. This requires crowd avatar rendering. Even in a standalone VR world, there may also be some independent computer controlled intelligent avatars running randomly. The system should have enough resources to handle crowd avatars.

A widely used technique to implement human-like behavior animation is physics-based animation, also called dynamic animation, which has a 3D polygonal geometric model representing the human-like avatar and a mathematical model describing the behavior. This complex avatar model is usually constructed with thousands of polygons, which consumes significant amount of graphics rendering resources. The physics-based movement mathematical model requires a significant amount of computation time. Furthermore, the human body detailed animations, such as muscle deformation, skin animation, and facial movement, will exhaust computer resources [1, 2, 7, 11]. Therefore, physical-based human-like behavior animation approaches are not apt to be employed in real-time VR worlds. Other approaches, such as kinematic approaches [3, 4] and keyframing techniques [5], suffer from similar problems and other difficulties. Because of the requirement of intensive calculation and expensive rendering, most of these approaches cannot achieve real-time in virtual environments.

Here we present an impostors-based approach to implement real-time human-like avatar behavior. Section 2 gives a brief overview of related work. Section 3 presents our approach followed by detail discussions in Section 4 and 5. Avatar related terrain following and

[*] This project has been funded by a grant from the National Science Foundation.

collision detection are discussed in Section 6. Finally, Section 7 summarizes our current and future work.

2. Related work

Maciel and Shirley introduced impostors-based technique to speed up 3D graphics scene rendering [6]. The basic idea of this technique is to use an image to represent a static complex 3D object from current viewpoint in the scene. This image is opaque where the object is present; everywhere else it is totally transparent. When the viewpoint moves within a certain adjacent area, this image can still approximately represent the object. Thus the system gains on performance by only mapping the image onto a rectangle (impostor), rather than rendering the complex 3D object, which may have thousands of triangles, or even curve surfaces. From this point of view, an impostor is a kind of rendering primitive that exploits frame-to-frame coherence efficiently if the viewer moves slowly. To be of any use, an impostor should be faster to draw than the object it represents, and it should closely resemble the object. Also of importance is that an impostor should be reused for several viewpoints located close together [15].

Given the view position and direction, how to find the right image of the object and map it onto the appropriate impostor with correct normal is the key of success for the impostors-based technique. There are two ways to prepare the image textures: dynamical on-the-fly generation and pre-generation. Maciel and Shirley have developed a navigation system of large environments that primarily relies on pre-generated impostors [6]. They replace clusters of objects by very simple texture-mapped geometry (impostors). Their system requires pre-generating *N* textures for every object, where *N* corresponds to the number of potential viewing directions. Schaulfer and Sturzlinger introduced the notion of dynamically generated impostors [8, 9]. When an object impostor needs to be dynamically generated, an off-screen buffer is set up by initializing its alpha channel to 0 (i.e., fully transparent). The object is then drawn into the buffer, and the alpha channel is set opaque (*alpha=1.0*) where the object is present. After the viewpoint is placed and oriented accordingly in this off-screen window, the impostor image is read back into the texture memory and used as a texture on the polygon that represents the object. Aubel and Boulic used Schaulfer's method to implement real-time display of virtual humans [10, 12]. Aliaga and others integrated the impostor techniques with portal culling algorithms to speed up walkthroughs of VR worlds [13, 14, 15]. Other solutions using impostors-based rendering/caching to accelerate walkthroughs of complex environments can be found in [16, 17, 18, 19]. Recent work on impostors-based techniques has focused on how to add depth information to achieve correct visibility, which is not the focus of this paper.

As we have seen, most impostor-related researches have been focusing on speeding up the walkthrough of 3D VR worlds. Their work so far has been mainly restricted to static scene and objects. Here we propose an approach that applies the impostor techniques to active objects, the avatars to achieve behavior human-like animation in VR worlds. Our approach is similar to but different from Aubel's method [10, 12]. We use *pre-captured real human movement pictures* as the impostors, while Aubel uses 3D model based dynamically generated textures. In Aubel system, physics-based 3D human-like models and behavior animation, which are the time-consuming and calculation-intensive, are still required. When a texture (impostor) needs to be updated, a snapshot of the 3D virtual human is taken from an off-screen frame-buffer. Aubel's system needs to take care of the complicated 3D virtual human rendering and movement calculation. It gains only from the frame-to-frame coherence property. Our approach does not need any 3D human-like model. Avatar behavior animations are totally based on the real human model pictures. Theoretically any behavior can be easily animated without costing much time and resources. Relying on the symmetry property of movement, the number of impostor images can be reduced by half.

3. General overview

It is well known that the human brain tends to interpolate and thus reconstruct the missing frames when shown only certain keyframes of a motion. This often makes a few key postures sufficient to understand an entire movement. This principle has been successfully used for decades in the cartoon industry. Our idea is to take the advantage of this principle and apply it to the avatar animation. The diagram of this approach is shown in Figure 1. For every behavior of each type of avatar, we take a minimum number of pictures. The number is chosen in such a way that these pictures can effectively and efficiently represent/describe the behavior, taking into account of the changes in the appearance of the avatar behavior. These pictures are then processed and converted into impostor images (usually with smaller sizes) described in following sections. These resulting impostor images are used as texture images during the behavior animation. The animation key task is to pick the right one from these pre-generated impostor images, and texture-map it onto a specific rectangle. This rectangle has appropriate size and is placed at right position with the normal direction always opposing the view direction. With the time elapsing and view direction changing, this polygon is dynamically placed to its new position and texture-mapped with new impostors.

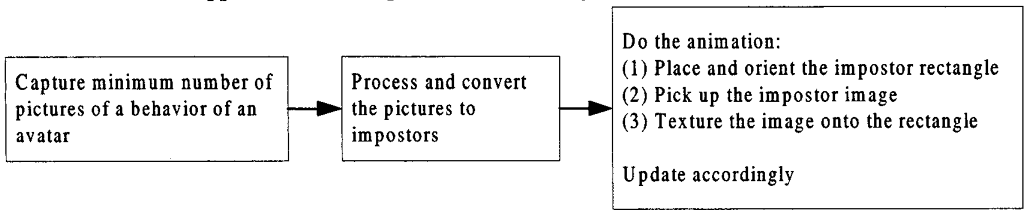

Capture minimum number of pictures of a behavior of an avatar
Process and convert the pictures to impostors
Do the animation:
(1) Place and orient the impostor rectangle
(2) Pick up the impostor image
(3) Texture the image onto the rectangle
Update accordingly

Fig. 1 Procedures of implementation

Therefore, we will have the avatar's smooth behavior if we have enough high resolution, which is decided together by the number of pictures taken in one direction within a cycle and the measure of angle between two neighboring directions. With this method, we have virtually decreased the polygon complexity of a human-like avatar model to a rectangle (Fig. 2). More importantly, we successfully skipped the complex physical-based simulation task. Very little calculation effort is needed.

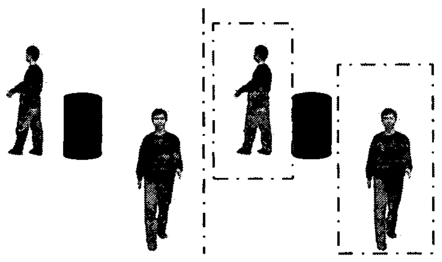


Fig. 2 Avatar models vs their impostors

In order to facilitate the discussion and description and without losing general generality, we take the complex walking behavior as the example in the following sections.

4. Image pre-generation

4.1 The symmetry principle of movement

Assume that an avatar has ***n*** different walking directions on the x-z plane (with positive y-axis in the direction down to the ground). For each direction, we need to capture ***m*** frame pictures within a walking cycle. These ***m*** frame pictures should effectively represent the walking movement. Now imagine that an avatar is walking toward positive x-axis (toward right), and you are standing at negative z-axis and looking toward positive z-axis (toward monitor). If you mirror what you see of the avatar movement around negative y-axis, you then see the movement that the avatar walks toward negative x-axis. Hence, if your walking direction is at alpha degrees north of positive x-axis, the mirror images of your movement are exactly the same as your movements toward the direction with alpha angle from the negative x-axis. This symmetry principle is demonstrated in Fig. 3. Relying on this principle, we can decrease the required impostor images for a behavior by almost half.

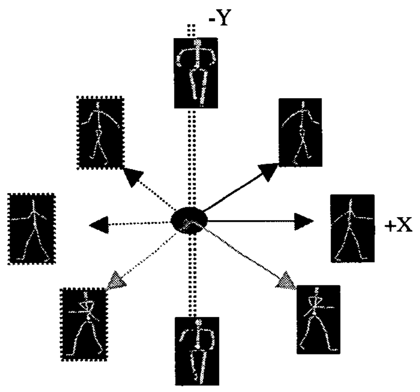


Fig. 3 Symmetry

4.2 Take behavior pictures

In order to guarantee that all the pictures are captured under exactly the same environment (e.g. with the same light sources and same size, thus same quality), we need to keep the distance between the human model and the camera constant. Also the focus and direction of camera should be kept static. This process is done in the following steps.

(1) Fix the camera at the origin facing positive z-axis. The model stands at the position 1-unit of the positive z-axis. The model walks without changing the position, just like walking on a treadmill machine. In order to facilitate this task, we can let the model walk on a treadmill that is on a turning table (Fig. 4);

(2) Decide the number of different directions between +90° and -90° (clockwise), say ***n*** different directions. The measurement of angle between any two neighboring directions should be the same. That is *180/(n-1)* degrees. For example, if *n=7*, then the seven walking directions are +90, +60, 30, 0, -30, -60, -90 (Fig. 4);

(3) Snapshot ***m*** frame of images within one cycle of the avatar walking movement for each direction (avatar walks without position changing). Then totally we have ***n*m*** frame of images.

After mirroring, these ***n*** direction images can serve as totally *2(n-1) + 2* different walking directions. For example, +150 direction is a mirror of +30 direction, and +180 a mirror of 0. As we mentioned above, ***n*** and ***m*** together decide the animation resolution. The bigger *n* and *m* means smoother animation result. For walking animation, ***m*** (frames per cycle) should be at least 4. Our experiments show that a walking animation with ***n****=7* and ***m****=8* yields a pleasing smooth walking animation in VR worlds.

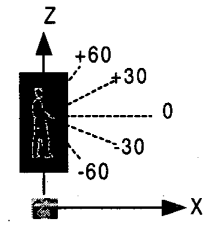


Fig. 4 Take pictures

4.3 Generate Impostors

After capturing those ***n*m*** images in the way described above, now we need to process and convert them into impostor (texture) images that meet the following requirement.

- **Image size:** Let ***w*** and ***h*** represent the width and height of an impostor image in the number of pixels, respectively. To achieve better animation performance, the values of both ***w*** and ***h*** should be the power of 2. Theoretically the ratio of ***w*** to ***h*** should be approximately equal to the ratio of height and width of the human model. But due to the texture mapping technique, we can only make choice among 32*16, 64*32, 128*64, 256*128, and so on for ***w*** and ***h*** values.
- **Transparency requirement:** With each impostor image, any pixel that doesn't belong to the model should be set to be totally transparent. Others must be set to be opaque.

Photoshop provides edge detection function to find the boundary of the model in the picture. Then, we can easy set those background pixels to be transparent. Photoshop also allows us to change image size conveniently. In WTK library, a black pixel (with R, G, and B are all 255) of a texture image is treated as full transparent if texture transparent function is enabled.

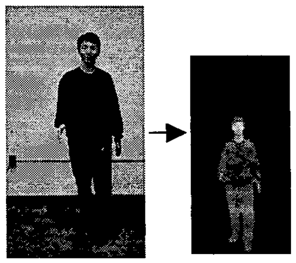


Fig. 5 Generate impostors

Figure 5 is an example of captured model picture and its impostor image (64*128) generated using Photoshop.

5. Avatar behavior animation in VR systems

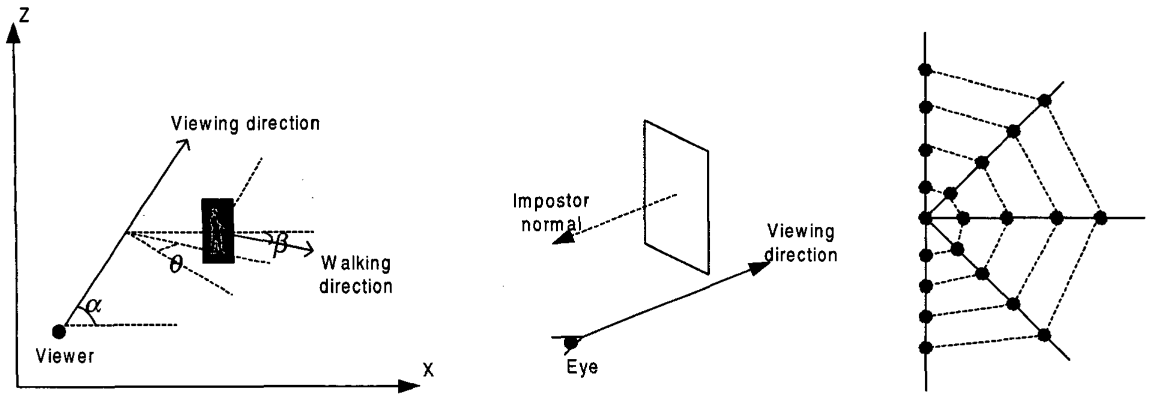
The main idea behind this method is *"placing appropriate impostor image at the right position"* within the 3D virtual world. When animating an avatar walking in VR worlds, we place/update the impostor according to the current view direction and the avatar position and orientation. A rectangle geometry object is created to represent an avatar (the impostor polygon of the avatar). The width and height of the rectangle should be approximately the same of the avatar.

(1) Place/translate the impostor rectangle to the new position where the avatar will be at the current time in the 3D world coordinate system in the VR world. Assume β represents the avatar walking direction, which is the angle between the avatar walking direction and the positive x-axis within the right-hand coordinate system, and *DeltaD* represents the avatar moving distance after a small animation time slice *(deltaT)*. The new position of the avatar in the world coordinate system is[1]:

$$X_{new} = X_{old} + DeltaD * \cos(\beta)$$
$$Z_{new} = Z_{old} + DeltaD * \sin(\beta)$$

(2) Orient the impostor rectangle. Orient the impostor rectangle so that it always faces the viewer. The normal of the rectangle is oriented to be the opposite of view direction (Fig. 6(b)).

[1] Here we assume the avatar is walking on a piece of flat floor. Thus the height component (Y) does not change while walking. Also notice that the new avatar position is independent of current viewpoint and view direction. In order to save calculation time, the *DeltaD*sin(β)* and *DeltaD*cos(β)* are calculated and the results are saved whenever the walking direction or/and view direction is/are changed, so that they become constants while walking forward/backward. On the other hand, the rectangle has the depth information.



(a) Determine the relative walking direction

(b) Place the impostor

(c) Animation resolution: 45 degrees, 5 frames/cycle

Fig. 6 Impostor-based animation

Talbel: Corresponding actions under different situations

		Avatar		
		Still	**Turn**	**Walk F/B**
View Point	**Still**	Do nothing	♦ Choose the closest direction under which impostors are pre-generated	♦ Translate impostor ♦ Update impostor image within the same walking direction
	Turn	♦ Choose the closest direction under which impostor images are pre-generated ♦ Orient impostor so that its normal is opposite to the view point direction	♦ Choose the closest direction under which impostors are pre-generated ♦ Orient impostor so that its normal is opposite to the view point direction	♦ Translate impostor ♦ Choose the closest direction under which impostors are pre-generated ♦ Update impostor image ♦ Orient impostor
	Move F/B	Do nothing	♦ Choose the closest direction under which impostors are pre-generated	♦ Translate impostor ♦ Update impostor image within the same walking direction

(3) Determine the avatar's walking direction in the viewing coordinate system. This decides from which set of impostor images the new imposter image will come from. Let α represent the viewing direction in world coordinate system, which is the angle between the viewing direction and the positive x-axis in the world coordinate system (Fig. 7(a)). Then the angle (θ) between the walking direction and the view direction in the viewing coordinate system is:

$$\theta = 90 - (\alpha - \beta)$$

Based on the θ value, choose a closest walking direction under which impostor images have been pre-generated (Fig. 6(c)).

(4) Choose the impostor image. Based on the sequence number of the previous used frame, pick up the next image frame. For example, if five frames are pre-generated for one cycle of walking, and labeled as 0, 1, 2, 3, and 4. If previous frame used is No. 1, the next frame to be used is No. 2 frame. If frame 4 is used currently, then next one is frame 0 (Fig. 6(c)).

(5) Texture-map the impostor image onto the impostor polygon.

(6) Finally, perform the terrain following and collision detection on the impostor (see Section 6 below).

Table 1 summary of all possible actions under different situations.

6. Avatar terrain following and collision detection

Avatar walks usually on some object, such as ground, floor, or box in VR worlds. Therefore performing terrain following on avatar is very important to increase realism. So does also for avatar collision detection, which prevents that the avatar goes through obstacles, such as walls and solid 3D objects. With the impostors-based method, we don't have 3D model avatars on which the traditional collision detection techniques work on. Our avatars are 2D impostors. We have to develop new methods to implement these tasks.

6.1 Terrain following

Performing terrain following on an impostor-based avatar is relatively easy. Assume that our avatar is far above the terrain, we can emit a ray in the down direction. This ray originates from the middle point of the bottom side of the impostor rectangle. The height of the first intersection point between the ray and any polygon in the world should be the (bottom side) height of the impostor. After we find the first intersection point, we only need to translate the impostor down so that the middle point on the bottom has the same coordinates of the intersection point. The result of this is that the impostor-based avatar is standing on the object (floor).

6.2 Collision detection

Many collision detection techniques have been developed and used in VR worlds. The first step of these techniques is to check out whether the two geometrical boundaries of the objects intersect with each other. If intersection doesn't exist, no collision between these two objects exists. Otherwise, further detection can be done. Obviously, a 3D geometrical boundary of an avatar has to be involved in order to perform collision detection between it and other 3D objects in VR worlds.

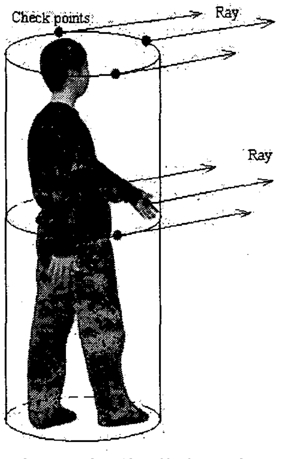


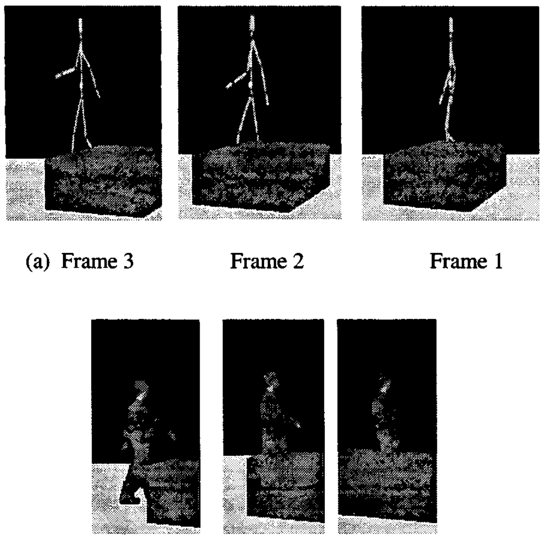
Fig. 7 Virtual cylinder and collision detection check points

In our approach, our avatars are 2D impostor images, which are geometry rectangles and have only 2D geometrical boundaries. Therefore we cannot use the existing collision detection techniques directly to perform 3D collision detection with 2D impostors. One way to use existing methods is to create an avatar-size 3D box for each avatar. The 3D box is used only by collision detection algorithm for detecting the possible collision between this avatar and other objects in VR worlds. The 3D box is so placed that the box center (middle point) is on the avatar impostor plane and overlaps with the center of the avatar impostor. When any intersection between the 3D box and any other object is detected, we assume that the avatar impostor collides with that object.

Another simple approximate method we implemented for collision detection on avatars is as follows. Imagine that a 3D avatar is surrounded by a virtual cylinder. The height of the cylinder equals to the avatar height, and its diameter is the minimum size of the avatar in horizontal plane, taking into account of all possible gestures. Before the avatar moves forward a *deltaD* distance, we check totally 6 points on the virtual cylinder for potential intersection with other objects in the VR world. Fig. 7 shows the locations of the six checking points. A ray is emitted from each checking point with the same direction of walking direction. A distance between a checking point and the first intersection point that the ray is intersected with any other object is then calculated. If the minimum of the six distances calculated is smaller than the *deltaD*, no collision will happen if the avatar moves forward a *deltaD* distance. Otherwise, collision is detected, and the minimum distance is also the maximum distance the avatar impostor is allowed to move forward. From the Fig. 7, we can see that if the height of an object is smaller than the half of the avatar's height, the avatar collision with the object will not be detected. This is reasonable because we can think that this lower object cannot block the avatar walking. With the terrain following implemented, the avatar will stand on the object. The advantages of this method are that no extra 3D avatar model is needed and very little resource is consumed. The main disadvantage is that it can only perform approximate collision detection.

7. Summary and future work

We have successfully implemented the presented approach on Windows95/98/NT platform in our Multi-User Virtual Environments system (MUVEs). MUVEs are PC-based networked virtual environments that use digitized museum resources to enhance middle school students' learning about science. It allows multi-user avatars simultaneously navigating VR worlds. Fig. 8 to Fig. 11 are some snapshots from MUVEs. Fig. 8 shows an avatar walking behind a red box. The feet are culled because of depth information associated with the impostor. Fig. 9, Fig. 10 and Fig. 11 show the impostors-



(a) Frame 3 Frame 2 Frame 1

(b) Frame 1 Frame 2 Frame 3

Fig. 8

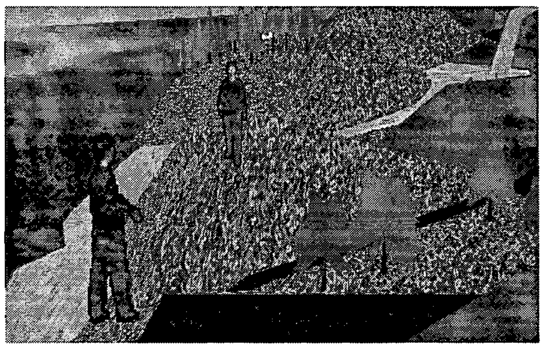


Fig. 9

based avatars walking in the River City and the Park of the MUVEs system.

The main advantages of our impostors-based method are:

- It can animate avatar behavior (walking, running, jumping, sport activity behaviors, and others) without any complex animation calculation. It saves calculation time!
- Use simple texture mapped rectangle to represent complicated 3D avatar model. It saves rendering time!

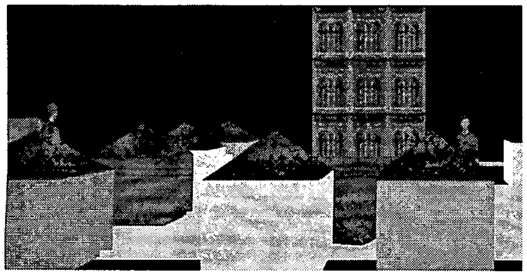


Fig. 10

- It is easy to achieve real human-like behavior animation in VR worlds. Your avatar may look like exactly what you are in the real world.
- This approach is also suited to other static but complicated 3D object, thus to speed up rendering.

The method's disadvantages include:

- It requires a pre-generating (processing) stage.
- It may suffer texture memory problem. If we cache all the currently used images in memory, we have to expand our computer's memory. If we don't cache the images and store them in hard disk, the importing of an image when it is required may take some time, even though the size of each image is very small (2KB for a 32*64 texture image). Thus limitation on the number of avatar types may occur.
- Animation quality may suffer from the possible low animation resolution. When the distance between the eye (viewpoint) and the avatar is too close, the avatar figure may look unrealistic.

We have presented a new method that uses impostors-based approach to achieve real-time human-like avatar behavior in virtual reality worlds. Experiment results show significant realism and efficiency improvement. Currently, we are working on a method that automatically generates impostor textures from snapped model pictures.

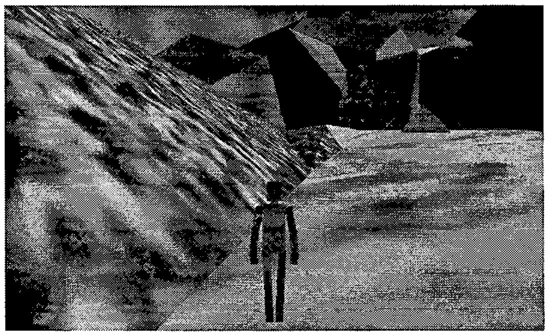


Fig. 11

References

[1] D. Brogan, R. Metoyer, and J. Hodgins, "Dynamically simulated characters in virtual environments," *IEEE Computer Graphics and Applications*, Vol. 18, 1998, pp.58-69

[2] J. Hodgins et al., "Animating human athletics," in *Computer Graphics (Proc. SIGGRAPH'95)*, R. Cook, ed., ACM Press, New York, 1995, pp.71-78

[3] R. Boulic, "A global human walking model with real-time kinematic personification", *The Visual Computer,* Vol.6, 1990, pp. 344-358

[4] H. Ko and N. Badler, "Straight-line walking animation based on kinematic generalization that perserves the original characteristics," *Graphics Interface'93*, pp. 9-16

[5] D. Sturman, "Interactive keyframing animation of 3-D articulated models," *Graphics Interface'86*, Tutorial on Computer Animation, 1986

[6] P. Maciel and P. Shirley, "Virtual navigation of large environments using textured clusters," in *Proc. 1995 Symp. Interactive 3D Graphics*, pp. 95-102

[7] S. Chung and J. Hahn, "Animation of human walking in virtual environments," in *Proceedings of Computer Animation*, 1999, pp. 4-15

[8] G. Schaulfer and W. Sturzlinger, "A three dimensional image cache for virtual reality," in *Proc. Eurographics'96*, pp. 227-234

[9] G. Schaufler, "Exploiting frame-to-frame coherence in a virtual reality system," in *IEEE Proc. Virtual Reality Annual International Symposium*, 1996, pp. 95 -102

[10] Aubel, R. Boulic, and D. Thalmann, "Real-time display of virtual humans: levels of details and impostors," *IEEE Transaction on Circuits and Systems for Video Technology,* Vol. 10, No.2, 2000, pp. 207-217

[11] T. Capin, I. Pandzic, and N. Thalmann, "Integration of avatars and autonomous virtual humans in networked virtual environments," http://www.miralab.unige.ch/index.html

[12] Aubel, R. Boulic, and D. Thalmann, "Animated impostors for real-time display of numerous virtual human," *in Proc. Virtual Worlds'98*, Paris, France, 1998, pp. 14-28

[13] D. Aliaga, J. Cohen, A. Wilson, etc, "A Framework for the real-time walkthrough of massive models," *Technique Report UNC TR#98-013*, Computer Science Department, University of North Carolina at Chapel Hill

[14] D. Aliaga and A. Lastra, "Architectural walkthroughs using portal textures," in *IEEE Visualization'97*, October 1997, pp. 355-362

[15] T. Moller and E. Haines, "Real-time rendering," *A K Peters, Ltd*, 1999

[16] X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey, "Multi-layered impostors for accelerated rendering," *in Proc. Eurographics'99, Computer Graphics Forum*, Vol. 18, 1999, pp. 61-72

[17] J. Airey, R. John, and B. Frederick, "Towards image realism with interactive update rates in complex virtual building environments," in *Computer Graphics Symposium on Interactive 3D Graphics*, Vol. 24, No. 2, 1990, pp. 41-50

[18] J. Shade, D. Lichinski, D. Salesin, etc. "Hierarchical image caching for accelerated walkthroughs of complex environments," *in Proc. Computer Graphics, SIGGRAPH'96*, pp. 75-82

[19] P. Ebbesmeyer, "Textured virtual walls - achieving interactive frame rates during walkthroughs of complex indoor environments," in *IEEE Proc. Virtual Reality Annual International Symposium*, 1998, pp. 220 -227