Animated Impostors for Real-time Display of Numerous Virtual Humans

Amaury Aubel, Ronan Boulic, Daniel Thalmann
Computer Graphics Lab
Swiss Federal Institute of Technology (EPFL)
CH 1015 Lausanne, Switzerland
Tel: +41 21 693 52 14 Fax: +41 21 693 53 28
e-mail: {aubel,boulic,thalmann}@lig.di.epfl.ch

Abstract

Rendering and animating in real-time a multitude of articulated characters presents a real challenge and few hardware systems are up to the task. Up to now little research has been conducted to tackle the issue of real-time rendering of numerous virtual humans. However, due to the growing interest in collaborative virtual environments the demand for numerous realistic avatars is becoming stronger.
This paper presents a hardware-independent technique that improves the display rate of animated characters by acting on the sole geometric and rendering information. We first review the acceleration techniques traditionally in use in computer graphics and highlight their suitability to articulated characters. Then we show how impostors can be used to render virtual humans. Finally we introduce concrete case studies that demonstrate the effectiveness of our approach.

Keywords: virtual humans, level-of-detail, impostors, image caching, image-based rendering.

1. Introduction

Even though our visual system is not deceived yet when confronted with virtual humans, our acceptance of virtual characters has greatly improved over the past few years. Today's virtual humans faithfully embody real participants in collaborative virtual environments for example and are even capable of conveying emotions through facial animation [1]. Therefore, the demand for realistic real-time virtual humans is becoming stronger. Yet, despite the ever-increasing power of graphics workstations, rendering and animating virtual humans remain a very expensive task. Even a very high-end graphics system can have trouble sustaining a sufficient frame rate when it has to render numerous moving human figures commonly made up of thousands of polygons. While there is little doubt that hardware systems will eventually be fast enough, a few simple yet powerful software techniques can be used to speed up rendering of virtual humans by an order of magnitude.

Because 3D chips were not affordable or did not even exist in the 80s, video game characters, human-like or not, were then represented with 2D sprites. A sprite can be thought of as a block of pixels and a mask. The pixels give the color information of the final 2D image while the mask corresponds to a binary transparency channel. Using sprites, a human figure could easily be integrated into the decor. As more computing power was available in the 90s, the video game industry shifted towards 3D. However, the notion of sprites can also be used in the context of 3D rendering. This has been successfully demonstrated with billboards, which are basically 3D sprites, used for rendering very complex objects like trees or plants. In our opinion image-based rendering can also be used in the case of virtual humans by relying on the intrinsic temporal coherence of the animation.

Current graphics systems rarely take advantage of temporal coherence during animation. Yet, changes from frame to frame in a static scene are typically very small, which can obviously be exploited [9]. This still holds true for moving objects such as virtual humans providing that the motion remains slow in comparison with the graphics frame rate. We present in this paper a software approach to accelerated rendering of moving, articulated characters, which could easily be extended to any moving and/or self-deforming object. Our method is based on impostors, a combination of traditional level-of-detail techniques and image-based rendering and relies on the principle of temporal coherence. It does not require special hardware (except texture mapping and Z-buffering capabilities, which are commonplace on high-end workstations nowadays) though fast texture paging and frame buffer texturing is desirable for optimal performance.

The next section gives an overview of the existing schemes that are used to yield significant rendering speedups. Section 3 briefly describes our virtual human model and discusses the limitations of geometry level-of-detail techniques. The notion of dynamically generated impostors is introduced in section 4 with the presentation of an algorithm that generates virtual humans from previous rasterized images. In section 5 the duration of the validity of the image cache is discussed. The following section then shows how this method can be embedded in a large-scale simulation of a human flow. Finally section 7 concludes the paper with a summary of the results we obtained and leads on to some possible future work.

2. Background

There exist various techniques to speed up the rendering of a geometrical scene. They roughly fall into three categories: culling, geometric level-of-detail and image-based rendering which encompasses the concept of image caching. They all have in common the idea of reducing the complexity of the scene while retaining its visual characteristics. Besides, they can often be combined to produce better results, as shown in sections 3 and 4.

Culling algorithms basically discard objects or parts of objects that are not visible in the final rendered image: an object is not sent to the graphics pipeline if it lies outside the viewing frustum (visibility culling) or if it is occluded by other parts of the scene (occlusion culling). Luebke and Georges described an occlusion culling system well suited for highly occluded architectural models that determines potentially visible sets at render-time [3]. One major drawback of occlusion culling algorithms is that they usually require a specific organization of the whole geometry database: the scene is typically divided into smaller units or cells to accelerate the culling process. Therefore, they perform poorly on individual objects. Because of this restriction it is no use trying to apply occlusion techniques to virtual humans if they are considered individually as we do. Nevertheless, occlusion culling might be contemplated at a higher level if the virtual humans are placed in a densely occluded world, as could be the case when simulating a human crowd for instance.

Geometric level of detail (LOD) attempts to reduce the number of rendered polygons by using several representations of decreasing complexity of an object. At each frame the appropriate model or resolution is selected. Typically the selection criterion is the distance to the viewer although the object motion is also taken into account (motion LOD) in some cases. The major hindrance to using LOD is related to the problem of multi-resolution modeling, that is to say the automatic generation from a 3D object of simpler, coarser 3D representations that bear as strong a resemblance as possible to the original object. Heckbert and Garland [4] presented a survey of mesh simplification techniques. More recently, some more work has been done to provide visually better approximations [5,6]. Apart from the problem of multi-resolution modeling, it is usually quite straightforward to use LODs in virtual reality applications. Funkhouser and Séquin described a benefit heuristic [15] to select the best resolution for each rendered object in the scene. Several factors are taken into account in their heuristic: image-space size so as to diminish the contribution of distant objects, distance to the line of sight because our visual system perceives with less accuracy objects on the periphery, motion because fast moving objects tend to appear blurred in our eyes. In addition, objects can be assigned different priorities since they all may play not an equal role in the simulation. Finally, hysteresis is also taken into consideration in order to avoid the visually distracting effect that occurs when the object keeps switching between two LODs. Recently Pratt et al. [14] have applied geometric LODs to virtual humans in large-scale, networked virtual environments. They used heuristic to determine the viewing distances that should trigger a LOD switch. Each simulated human has four different resolutions ranging from about 500 polygons down to only three polygons. However, their lowest resolutions are probably too coarse to be actually used unless the virtual human is only a few pixels high, in which case another technique than polygon rendering may be considered.

Finally, image caching is based on the principle of temporal coherence during animation. The basic idea is to re-use parts of the frame buffer content over several frames, thus avoiding rendering for each frame the whole scene from scratch [2]. The hardware architecture Talisman [7] exploits this temporal coherence of the frame buffer: independent objects are rendered at different rates into separate image layers which are composited together at video rate to create the displayed image. In addition, the software host is allowed to maintain a priority list for the image layers, which means that objects in the foreground can be updated more frequently than remote objects in the background. From a software point of view Maciel et al. [8] presented a navigation system of large environments that primarily relies on impostors. They replaced clusters of objects with texture-mapped planes, one of the first attempts at software image-based rendering. But they do not generate the textures on the fly. Hence their system requires a pre-processing stage during which the scene is parsed: N textures - where N corresponds to the number of potential viewing directions - are generated for every object and stored in memory. This approach is thereby likely to consume rapidly the texture memory. Shaufler and Stürzlinger first introduced the notion of dynamically generated impostors [9] that correspond essentially to a software image cache. They managed to solve the problem of texture memory usage since textures are generated on demand in their system. Since then two other software solutions using image-based rendering have been proposed concurrently and independently to accelerate walkthroughs of complex environments [10,11].

Although impostors appear to be one of the most promising techniques their use has mainly been restricted so far to walkthroughs of large environments. We believe however that this technique can bring a substantial improvement to virtual humans too.

3. A multi-resolution virtual human

In this section we show how a multi-resolution virtual human can successfully be constructed and animated. We have designed a human modeler that solves the traditional multi-resolution problem described above. By relying on implicit surfaces we avoid resorting to complicated algorithms and automatically generate multiple look-alike resolutions of a model.

3.1. Multi-resolution modeling

Our real-time virtual human model consists of an invisible skeleton and a skin. The underlying skeleton is a hierarchy of joints that correspond to the real human main joints. Each joint has a set of degrees of freedom, rotation and/or translation, which are constrained to authorized values based on real human mobility capabilities [12]. Unlike other attempts [14] we did not model a multi-resolution skeleton because our purpose was not to demonstrate the effectiveness of animation level-of-detail. Hand joints can be replaced with a single joint though.

The skin of our virtual human is generated by means of an in-house modeler that relies on a multi-layer approach (fig. 1): ellipsoidal metaballs and ellipsoids are employed to represent the gross shape of bone, muscle and fat tissue [13]. Each primitive is attached to its proximal joint in the underlying human skeleton. The set of primitives then defines an implicit surface that approximates the real human skin. Sampling this implicit surface results in a polygonal mesh, which can be directly used for rendering.

Sampling the implicit surface is done as follows: we start by defining contours circling around each limb link of the underlying skeleton. We then cast rays in a star-shaped manner for each contour, with ray origins sitting on the skeleton link. For each ray, we compute the outermost intersection point with the implicit surface surrounding the link. The intersection is a sample point on the cross-section contour. Once all the sampling is done it is quite straightforward to construct a mesh from the sample points. Thus, we can generate more or less detailed polygonal meshes by simply varying the sample density i.e. the number of sampled contours as well as the number of sampled points per contour. And yet, since the implicit surface remains the same no matter the sampling frequency, the generated meshes look very similar (fig. 2).

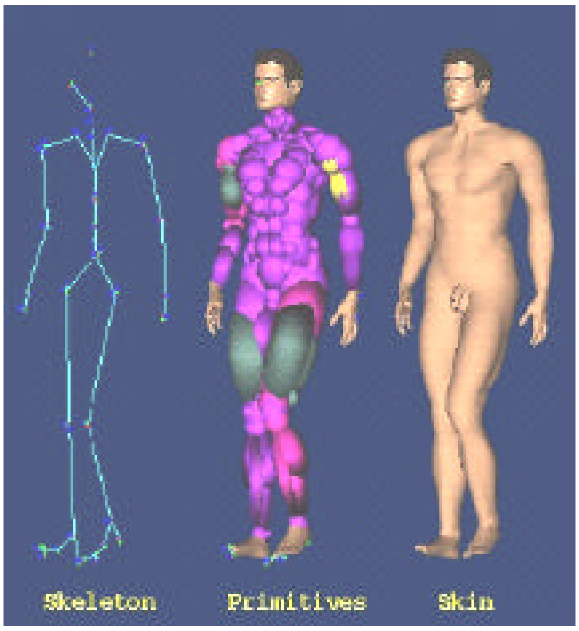


Figure 1: multi-layered model

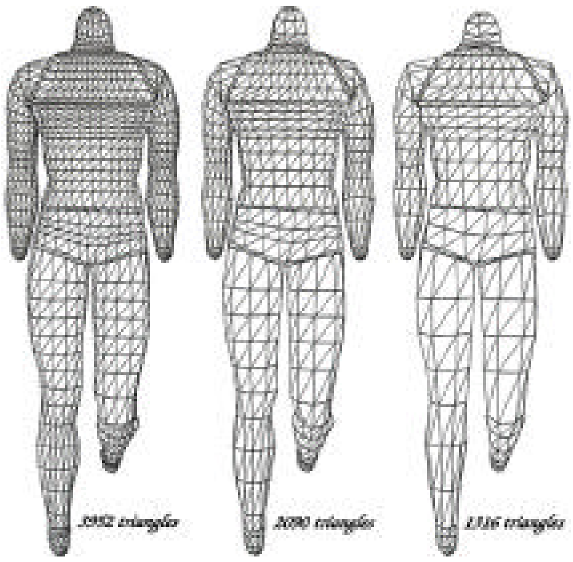


Figure 2: Body meshes of decreasing complexity from one implicit surface

As for the head, hands and feet, we still have to rely on a traditional decimation technique to simplify the original mesh. Manual intervention is still needed at the end of this process to smooth the transition between LODs. The body extremities can cleverly be replaced with simple textured geometry for the lowest resolution (fig. 3), which dramatically cuts down the number of triangles.

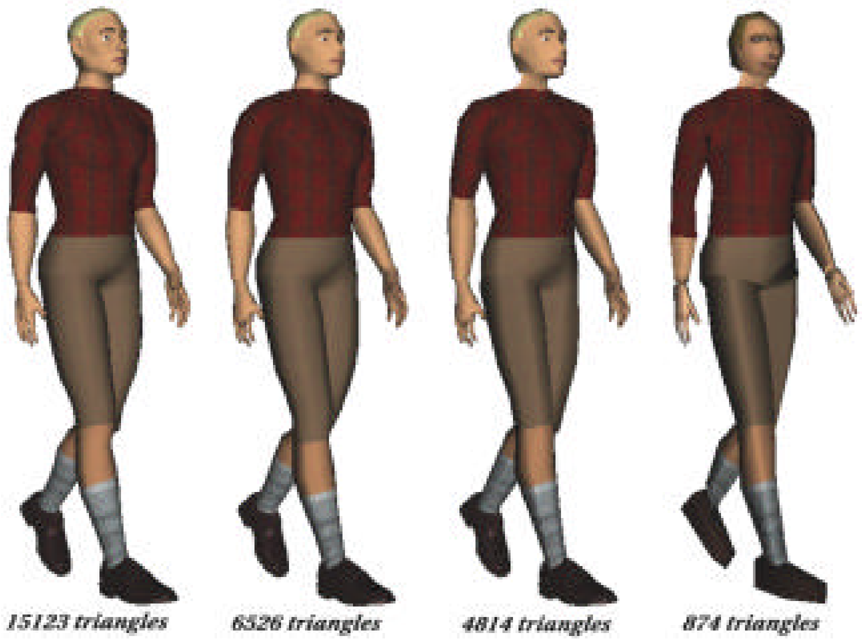


Figure 3: Levels of Details

3.2. Animation

The skeleton of our virtual human comprises a total of 74 DOFs corresponding to the real human joints plus a few global mobility nodes, which are used to orient and position the virtual human in the world. In the broad lines, animating a virtual human consists in updating this skeleton hierarchy including the global mobility joints at a fixed frame rate. There exist several techniques to feed the joints with new angle/position values. Motion capture for instance is used to record the body joint values for a given lapse of time. The motion can later be played back on demand for any virtual human. Key frame animation is another popular technique in which the animator explicitly specifies the kinematics by supplying key-frame values and lets the computer interpolate the values for the in-between frames.

During animation the appropriate resolution is selected for each individual according to the euclidian distance to the viewpoint. Therefore far objects and those on the periphery contribute less to the final image. Note that we could also take the general motion of the virtual human into account. Finally at a higher level, typically the application layer, virtual humans could be assigned different rendering priorities too. For example, in the context of a human crowd simulation where individuals move and act in clusters the application could decide to privilege some groups.

4. Animated Impostors for Articulated Characters

As first defined by Maciel et al. [8] an impostor is in essence some very simple geometry that manages to fool the viewer. As traditionally the case in the existing literature, what we mean by impostor is a set of transparent polygons onto which we map a meaningful, opaque image. More specifically our impostor corresponding to a virtual human is a simple textured plane that rotates to face continuously the viewer. The image or texture that is mapped onto this plane is merely a “snapshot” of the virtual human. Under these conditions if we take for granted that the picture of the virtual human can be re-used over several frames then we have virtually decreased the polygon complexity of a human character to a single plane (fig. 4).

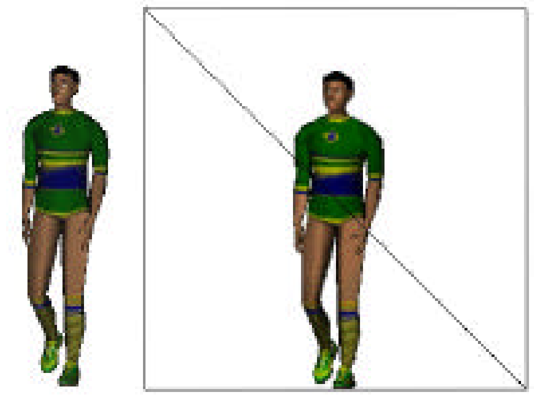


Figure 4: A Brazilian football player and its impostor

4.1. Impostor Refreshment Approach

The texture that is mapped onto the transparent plane needs to be refreshed from time to time because of the virtual human’s mobility or camera motion. Whenever the texture needs to be updated a snapshot of the virtual human is taken. This process is done in three steps:

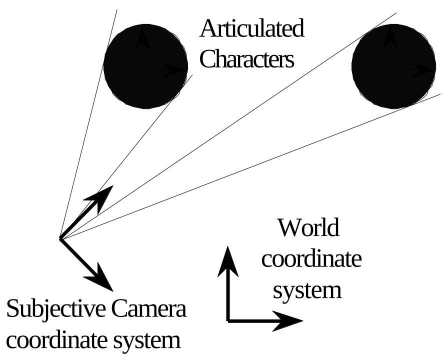
1) Set up an off-screen buffer that will receive the snapshot
2) Place the virtual human in front of the camera in the right posture
3) Render the actor and copy the result into texture memory

The first stage typically comes down to clearing the buffer. The buffer should be a part of the frame buffer so as to benefit from hardware acceleration. The purpose of the second step is to set up the proper view to take a picture: first, we let the actor strike the right pose depending on joint values. Second, it is moved in front of the camera or alternately, the camera itself is moved. Note that the latter is preferable because the skeleton hierarchy remains unchanged in this case, which may save some time. In the last stage the virtual human is rendered and the resulting image is copied into texture memory. Once the texture has been generated there only remains to render the textured billboard to have a virtual human on screen. The whole process is hardly any slower than rendering the actual 3D geometry of the virtual human for the following reasons: setting the off-screen buffer up can be done once for all in a pre-processing step. It chiefly consists in adjusting the viewing frustum to the virtual human. Furthermore, this buffer can be re-used to generate several textures of different actors providing that they have approximately the same size. Clearing the buffer is definitely not a costly operation. Letting the virtual human assume the right posture and rendering it would also have to be carried out if the real geometry was used instead. Finally, if the hardware features frame buffer texturing, the frame buffer to texture memory copy is performed within no time. If not, clearing the off-screen buffer and transferring its content still remain all the less costly that the buffer size is small, typically 128 by 128 pixels or less. As a consequence even in the worst cases (very high texture refreshment rates), impostors prove not to be slower than rendering the actual 3D geometry.

4.2. Impostor Viewing and Projection

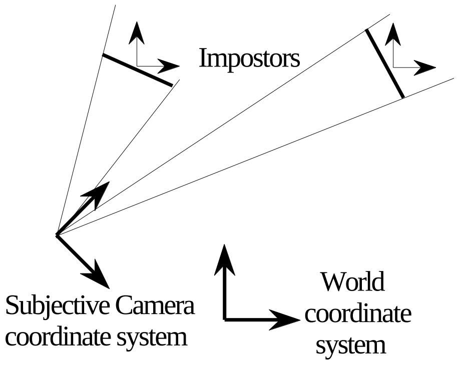
For the sake of clarity, when speaking respectively in the following of texture windows and scene window, we will mean the off-screen buffers where textures of virtual humans are generated and the window where the final, global scene with impostors is rendered.

a *Articulated characters rendering*



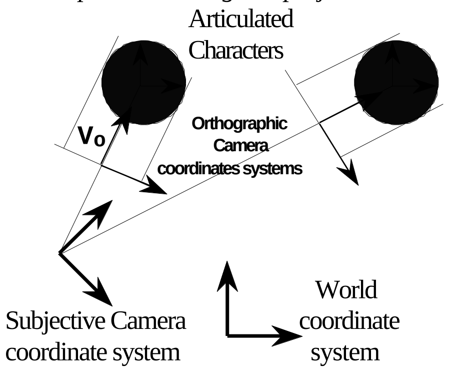
Drawing a. represents a top view of a scene containing two articulated characters. Each character has its own coordinates system. The subjective camera coordinate system defines the user's viewpoint and corresponds to the camera in the scene window. All coordinates systems are expressed with respect to the world.

b *Impostors rendering*



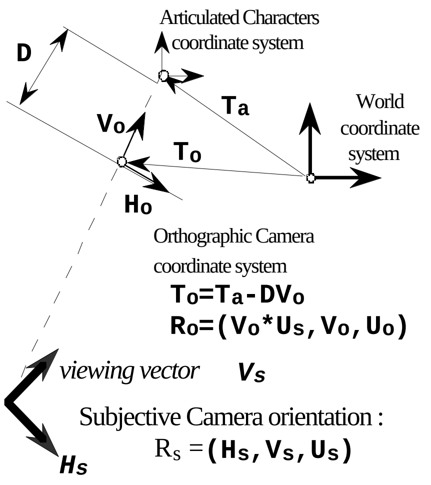
Drawing b. shows a top view of the same scene rendered with impostors. Each character's geometry is replaced with a single textured plane. Each plane is oriented so that its normal vector points back to the eye i.e. the origin of the subjective camera coordinate system.

c *Impostors viewing and projection*


Articulated Characters
V_o
Orthographic Camera coordinates systems
Subjective Camera coordinate system
World coordinate system

Drawing c. explains how the textures mapped onto the planes in *drawing b.* are generated i.e. how snapshots of virtual humans are taken. In the drawing there are two orthographic camera coordinates systems corresponding to the cameras in the texture windows associated with each articulated character. The viewing direction **Vo** of the orthographic camera is determined by the eye's position and a fixed point (later referred to as **Ta**) of the articulated character, as will be explained.

d *Building the vi ewing transformation*



Articulated Characters coordinate system
D
Ta
World coordinate system
Vo
To
Ho
Orthographic Camera coordinate system
To=Ta-DVo
Ro=(Vo*Us,Vo,Uo)
viewing vector Vs
Subjective Camera orientation :
Rs =(Hs,Vs,Us)
Hs

Drawing d. is a more detailed version of the previous drawing. It mainly shows how the orthographic camera coordinate system is constructed. **Vo** is the unit vector derived from **Ta** and the position of the subjective camera. The two other unit vectors that give the orthographic camera's orientation **Ro** are determined as follows: the first vector **Ho** is the cross product of **Vo** and **Us** where **Us** is the normalized up vector of the subjective camera's frame. The cross product of **Ho** and **Vo** then yields **Uo**. Note that this method ensures we obtain a correct frame (**Ho≠0**) as long as the field of view is smaller than a half sphere.

Finally, the camera of the texture window (termed orthographic camera so far) must be positioned so that the virtual human is always entirely visible and of a constant global size when we take a snapshot. We proceed in two steps to do so. First, an orthographic projection is used because it incurs no distortion and actual sizes of objects are maintained when they are projected. Second, a fixed point named **Ta** is chosen in the body (it corresponds to the spine base**)** so that it is always projected onto the middle of the texture window. Practically, **Ta** is determined to be the "middle" point of the virtual human assuming a standing posture in which its arms and hands are fully stretched above its head. Note that this posture is the one in which the virtual human's apparent size is maximal. The orthographic camera is placed at distance D from **Ta** (D is a scalar). During the simulation, the camera is then moved at **To = Ta –** D.**Vo**. Hence **Ta**'s projection necessarily coincides with the center of the texture window. On account of the orthographic projection, D actually does not influence the virtual human's projection size. On the other hand the viewing frustum does. Consequently, it must be carefully chosen so that no parts of the virtual character are culled away. This is done in a pre-processing stage in which the viewing frustum's dimensions (in the texture window) are set to the maximal size of the virtual human (i.e. with arms raised above its head).

4.3. Several Texture Resolutions

All the work has been realized on Silicon Graphics workstations using the Performer graphics toolkit. Because of hardware/software restrictions concerning textures, texture windows have dimensions that are powers of two. Several texture windows of decreasing size can be associated with every virtual human. As the virtual human moves farther, which leads to smaller image-space size in the scene window, a smaller texture window can be used thus reducing texture memory consumption. Practically, we allocated 256x256, 128x128, 64x64 and 32x32 pixel texture windows for a simulation running at a 1280x1024 resolution. The largest windows were only used for close-up views. As always, it can be a bit difficult to strike the right balance between texture memory consumption and visual realism. In addition to better managing texture memory, using several texture windows of decreasing size also helps to reduce visual artifacts. As a matter of fact large textures of virtual humans that are mapped onto very small (far) planes might shimmer and flash as the impostors move. When several texture resolutions are employed, these artifacts tend to disappear because the appropriate texture resolution is selected according to the impostor image-space size, thus mimicking the well-known effect obtained with "mip mapping". Finally, when a texture is generated the appropriate LOD of the virtual human can be chosen based on the texture window size.

4.4. Main current limitation

Replacing a polygonal model of a virtual human with a single textured plane may introduce visibility problems: depth values of the texels are unlikely to match those of the actual geometry, which may lead to incorrect visibility, as illustrated in figure 5. We did not address this issue in this paper.

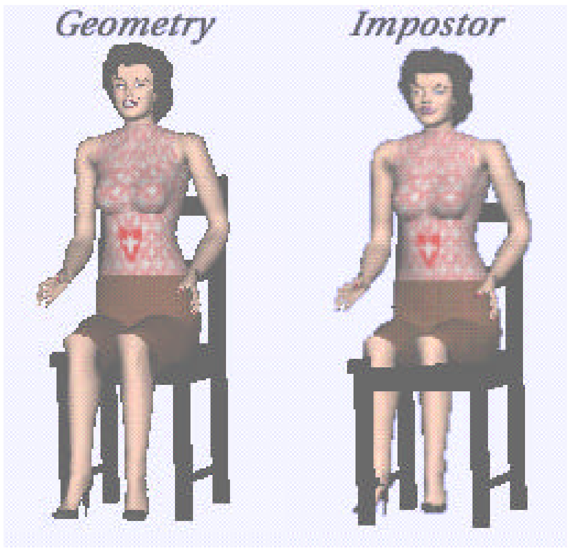


Figure 5: Incorrect Visibility

5. Duration of the image cache

For static objects camera motion is the one and only factor to take into consideration for cache invalidation [9,11]. The algorithm that decides whether a snapshot of a virtual human is stale or not is obviously a bit more complex. However, that algorithm has to execute very quickly because it must be performed for every virtual human at each frame. In our approach we distinguish two main factors: self-deformation of the virtual actor and its global motion with respect to the camera.

5.1. Virtual humans as self-deforming objects

Virtual humans can be considered as self-deforming objects. The basic idea for a cache invalidation algorithm is that the viewer need not see every posture of the virtual humans to fully understand what they are doing. A few key postures are often meaningful enough. We propose a simple algorithm that reflects the idea of sub-sampling the motion. The idea is to test distance variations between some pre-selected points in the skeleton. Using this scheme the virtual human is re-rendered if and only if the posture has changed significantly.

Once we have updated the skeleton hierarchy we have direct access to joints' positions in world space. It is therefore quite straightforward to compute distances (or rather squared distances to avoid unnecessary square roots) between some particular joints. In concrete terms we compare four distances (figure 6) with those stored when the texture was last generated. As soon as the variation exceeds a certain threshold the texture is to be re-generated. Of course the thresholds depend on the precision the viewer demands. Furthermore, length variations are weighted with the distance to the viewer to decrease the texture refreshment rate as the virtual human moves away from the viewpoint.

We found out that four tests suffice to reflect any significant change in the virtual human's posture. Nevertheless, other tests should be performed if the simulation is meant to underscore some peculiar actions e.g. grasping of an object requires additional testing of the hand motion. Similarly, it might be necessary to increase or decrease the number of tests when using non-human characters.

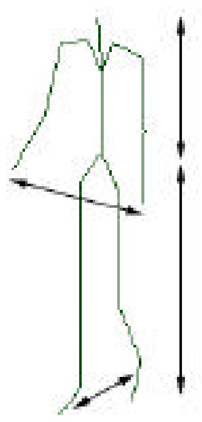


Figure 6: posture variations to test

5.2. Relative motion of the actor in the camera frame

Instead of testing independently camera motion and actor's orientation we have come up with a simple algorithm that checks both in a single test. The algorithm's main idea stems from the fact that every virtual human is always seen under a certain viewing angle which varies during simulation whether because of camera motion or actor's motion. Yet, there is basically no need to know what factor actually caused the variation.

In practice we test the variation of a "view" matrix, which corresponds to the transformation under which the viewer sees the virtual human. This matrix is plainly the product of the subjective camera matrix (camera in the scene window) and that of the articulated character. The cache invalidation algorithm runs in pseudo-code as follows:

```
For every virtual human at frame 0     /* Initialization stage */
    Generate texture
    Store View Matrix
End for

For every virtual human at frame N>0   /* Simulation */
    Compute new View Matrix
    Compute View Variation Matrix M (from previously stored view matrix to the newly computed one).
    Build axis/angle representation from the 3x3-rotation sub-matrix of M
    If angle>threshold
        Generate texture
        Store current View Matrix
    End if
End for
```

Building the axis-angle representation of a rotation matrix **M** consists in finding its equivalent rotation axis as well as the angle to rotate about it [16]. In practice, computing the rotation axis can be skipped since only the angle is needed afterwards. Finally, the threshold that indicates when the texture is to be regenerated can be weighted once again with the euclidian distance from the impostor to the viewer.

6. The human flow test bed

Our work on impostors originates from earlier research on human crowd simulation that was carried out in our laboratory. However, simulating a human crowd introduces many parameters that alter the frame rate results. We preferred to use a simpler environment in order to assess reliably the gain of impostors on geometry. In our simulation twenty walking virtual humans keep circling. They all move at different yet constant speeds, along more or less long circles (figure 7). A fast walking engine handles the motion of the actors, collision between characters are not detected and finally every articulated character always lies in the field of vision so that visibility culling does not play a role.

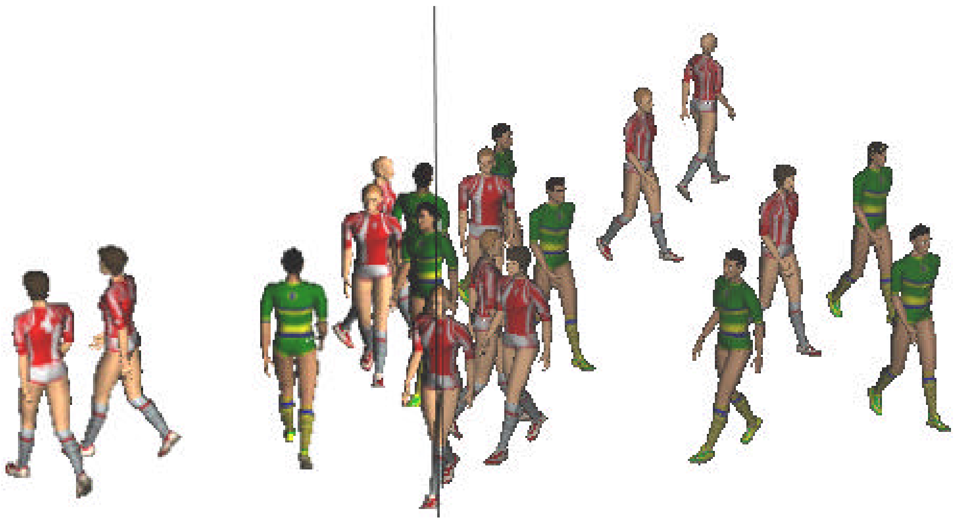
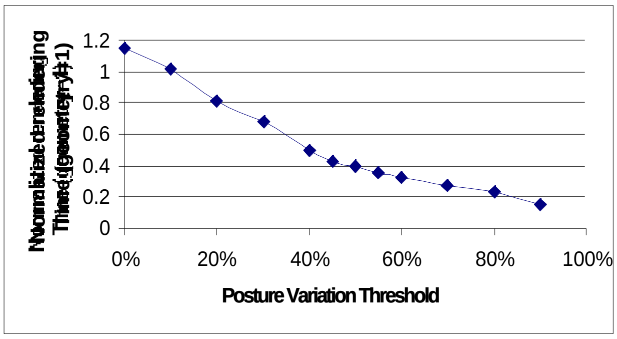
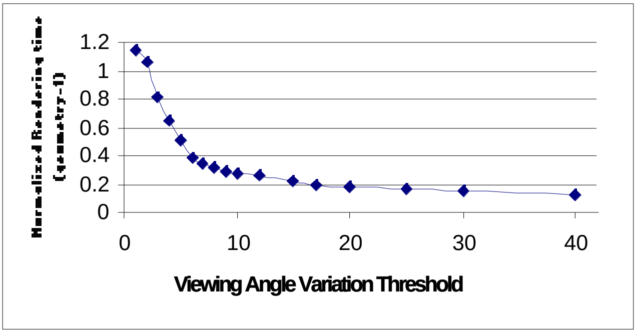


Figure 7: Simulating two football teams (left: impostors, right: real geometry)

The first graph shows the influence of the posture variation thresholds used in the texture cache invalidation mechanism. The other factor, that is the viewing angle, was deactivated throughout this test. Along the horizontal axis are noted the posture variation thresholds beyond which a texture is re-generated while the vertical axis shows the amount of time required for rendering the complete scene. The vertical axis is normalized with respect to the amount of time needed to render the whole scene using the real geometry. When the threshold is set to zero every change in the posture triggers a re-computation of the texture, in which case the rendering time logically exceeds the reference rendering time. Note that there is only a marginal difference of 15% though, which clearly shows that impostors are hardly slower than the actual geometry even in the worst cases. Rendering time plummets when the variation threshold is increased: a threshold of 40%, 60% and 80% cuts respectively by two, three and five the rendering time. In practice there is no noticeable difference in the animation of the actors as long as the threshold is smaller than 15%. However, it makes perfect sense to set the threshold to a much higher limit (between 40% and 80%) because the motion remains absolutely understandable. On the other hand it becomes hard to grasp that the actors are walking beyond 90%.

The second test focuses on the impact of the viewing angle on the rendering time. Like previously, the vertical axis is normalized with respect to a reference rendering time (that needed to render the whole scene with the actual geometry). On the horizontal axis are marked the viewing angle thresholds in degrees. Similarly to the first test we disabled the other factor in the cache invalidation mechanism. The critical threshold is reached for a viewing angle variation of two degrees only. Viewing angle thresholds of 5, 8 and 17 degrees cut respectively by two, three and five the rendering time. We consider that there is no real degradation of the animation up to 20 degrees. Refreshment of the actors' texture becomes too obvious beyond 30 degrees.





Graph 1&2: rendering speedups measurements

7. Conclusion

This paper shows that image-based rendering can be applied successfully to virtual humans. In particular, we have explained how to generate a texture representing a virtual human and proposed an image cache invalidation algorithm that works for any articulated character and executes reasonably fast. Two major issues, which were beyond the scope of this paper, could be addressed in future work:

1. Our texture refreshment algorithm performs on an individual basis. From the application point of view, the image cache invalidation mechanism should also consider virtual characters as a whole. Above all this would help achieve a quasi-constant frame rate, which is often regarded in virtual reality applications as more important than a high peak frame rate.

2. Because depth information is lost when complex 3D geometry is replaced with an impostor, especially with a plane, visibility may not be correctly resolved. Schaufler recently showed how to correct the visibility by directly modifying the depth value for every texel [17]. However, faster techniques could be investigated in the specific case of virtual humans. For example, the concrete problem depicted in figure 5 could be solved by decomposing the impostor's plane into several planes e.g. one for each major body part.

References

[1] T. K. Capin, I. Pandzic, H. Noser, N. Magnenat-Thalmann, D. Thalmann, "Virtual Human Representation and Communication in VLNET". In *IEEE Computer Graphics and Applications*, Vol. 17, No. 2, March - April 1997, pp. 42-53

[2] M.Regan, R.Post, "Priority Rendering with a Virtual Reality Address Recalculation Pipeline". In *Computer Graphics (SIGGRAPH '94 Proceedings)* pp. 155-162.

[3] F.Sillon, G.Drettakis, B.Bodelet "Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery". In *Proceedings of Eurographics'97*, 1997, pp. C-207 - C-217.

[4] P. Heckbert, M. Garland, "Muliresolution modelling for fast rendering". In *Proceedings of Graphics Interface'94*, pp. 43-50.

[5] J. Popovic, H. Hoppe, "Progressive Simplicial Complexes". In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pp. 217-224.

[6] M. Garland, P.S. Heckbert, "Surface Simplification Using Quadric Error Metrics". In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pp. 209-216.

[7] J. Torborg, J.T. Kajiya, "Talisman: Commodity Real-time 3D Graphics for the PC". In *Computer Graphics (SIGGRAPH '96 Proceedings)*, pp. 353-363.

[8] P.W.C. Maciel, P. Shirley, "Visual Navigation of Large Environments using Textured Clusters". In *1995 Symposium on Interactive 3D Graphics*, pp. 95–102

[9] G. Schaufler, W. Stürzlinger, "A Three Dimensional Image Cache for virtual reality". In *Proceedings of Eurographics'96*, pp. C-227 – C-234

[10] J. Shade, D. Lichinski, D.H. Salesin, T. DeRose, J. Snyder, "Hierarchical Image Caching for Accelerated Walthroughs of Complex Environments". In *Computer Graphics (SIGGRAPH'96 Proceedings)*, pp. 75-82.

[11] G. Schaufler, "Exploiting Frame to Frame Coherence in a Virtual Reality System". In *Proceedings of VRAIS'96*, Santa Cruz, California (April 1996), pp.95-102.

[12] R. Boulic., T.K. Capin, Z. Huang, P. Kalra., B. Lintermann., N. Magnenat-Thalmann, L. Moccozet, T. Molet, I. Pandzic, K. Saar, A. Schmitt, J. Shen, D. Thalmann, "The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters". In *Proceedings of Eurographics'95*, Maastricht, August 1995, pp. 337-348.

[13] J. Shen, E. Chauvineau, D. Thalmann, “Fast Realistic Human Body Deformations for Animation and VR Applications”. In *Computer Graphics International'96*, Pohang,Korea, June 1996, pp. 166-173.

[14] D.R. Pratt, S.M. Pratt, Paul T. Barham, R.E. Barker, M.S. Waldrop, J.F. Ehlert, C.A.Chrislip, “Humans in Large-scale, Networked Virtual Environments”. In *Presence*, Vol. 6, No. 5, October 1997, pp. 547-564.

[15] T. A. Funkhouser, C.H. Séquin, “Adaptative display algorithm for interactive frame rates during visualization of complex virtual environments”. In *Computer Graphics (SIGGRAPH’93 Proceedings)*, pp. 247-254.

[16] M. E. Pique “Converting between Matrix and Axis-Amount representations”. In *Graphics Gems (Vol. 1)*, Academic Press, pp. 466-467.

[17] G. Schaufler, “Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes”. In *Proceedings of 8th Eurographics workshop’97*. St. Etienne, France, June 16-18, 1997, pp. 151-162.