Volume 24 (2005), number 2 pp. 203–215

COMPUTER GRAPHICS *forum*

Survey of Real-Time Rendering Techniques for Crowds

G. Ryder and A. M. Day

School of Computing Sciences, University of East Anglia, U.K.

Abstract

Real-time rendering of photo-realistic humans is considerably outside the scope of current consumer-level computer hardware. There are many techniques, which attempt to bridge the gap between what is desired and what is possible. This paper aims to give an overview of the techniques designed to alter the complexity of the model's geometry (level of detail), or replace it with a flat image (visual impostor) and to improve the lighting model (lighting and shadows). Recent years have shown a boom in the power and availability of consumer-level programmable graphics processors, thus techniques that make use of these features are coming to the forefront.

Keywords: crowds, real-time, survey, virtual, human, rendering

ACM CCS: I.3.7 Computer Graphics: Three-dimensional graphics and realism, I.3.3 Computer Graphics: Picture/image generation

1. Introduction

The desire to render the world around us is common to many applications. As part of this 'virtual world' it is natural to include representations of humans. A realistic visual approximation of a human requires a polygon mesh of significant complexity, especially as it has to be sufficiently tessellated to allow smooth deformations of the model.

Visualizing one virtual human in real-time can consume a high proportion of processing time. In many situations it is not one, but a crowd of humans, which need to be rendered. The common aim of all of the papers discussed here is to reduce the computational expense of a particular element of the avatar, allowing interactive frame rates. In addition to this, the aim is to free the CPU to handle other elements of the simulation, e.g. A.I., I/O, or physics.

The current growth in power of consumer-level graphics processors is significant and with this many features are provided which are designed to facilitate efficient rendering of scene. As the growth in power continues, so does the desire to render even more complex scenes with realistic lighting. Visualizing virtual environments is a huge area of research, with many inter-related fields. However, this paper aims to cover only recent advances in the field applicable to the visualization of crowds of virtual humans.

What particular problems need to be overcome in visualizing virtual humans? The human body needs to be modeled as a deformable structure, thus many of the traditional techniques that are designed for rigid body techniques are not applicable. One partial solution is to split the human body into a series of rigid elements, considering each element as an individual body on which to apply the techniques. As many comment, including Aubel *et al.* [1], this causes problems at the joints (the boundaries between the rigid elements) that have to be carefully maintained to avoid cracks appearing. As with all deformable body work the mesh may be sufficiently tessellated in its rest pose. However, as the character moves and bends the model may be too coarse to give realistic deformation of joints.

Virtual humans also have further complications over standard deformable bodies. The human visual system puts a large emphasis on the human face and thus any interaction with a virtual human should be echoed in facial expression as well as the actions. Realistic lighting and self-shadowing contribute heavily to how we see facial expressions. This is very computationally expensive and has only recently been contemplated for real-time circumstances.

Submitted May 2004
Revised November 2004
Accepted April 2005

In this paper, level of detail (LOD) methods are first examined, looking at mesh decimation and mesh refinement techniques. Image-based rendering (IBR) techniques are examined for rendering crowds, and finally real-time shadow and lighting techniques are examined. Before examining the techniques, a quick overview of traditional human animation is given. It is useful to have an understanding of some popular methods for the animation of human meshes before considering how to increase the rendering speed of the scene. Occlusion is another large area, of which little has been focused directly on the rendering of crowds. However, Cohen *et al.* [2] present an excellent survey of visibility methods.

1.1. NURBS and Polygon Mesh Representation

The techniques in this paper concentrate on polygon representation of human models. However, the use of NURBS (nonuniform rational B-splines) is also common in the modeling of virtual humans. NURBS have some advantages over polygon meshes, their definition is more compact and natively allows infinite smoothness. However, they are significantly more complex to handle due to topology and other constraints. To render NURBS on traditional systems they must be passed to the graphics API as a polygon list, which negates much of the advantage of their description. A discussion of the problems with using a patchwork of NURBS in human animation is presented by Derose *et al.* [3]. Within the paper an alternative is given in the form of a "Hybrid Subdivision" method, which allows the incorporation of sharp and semi-sharp features as well as smooth features of a mesh. However, the method was never designed to be real-time and thus no indication of the runtime performance of the implementation is provided.

The methods reviewed in this paper are almost all solely based on the polygon mesh representation systems. The majority of the work on rendering virtual humans in real-time has concentrated on polygon mesh representations due to the fact that consumer-level hardware is optimized to handle such representations. The performance hit of not using the modern graphics cards hardware acceleration is high. Thus rendering techniques which lend themselves to implementation, either partial or fully, on the graphics processor unit (GPU) are in the forefront. Whether this course of hardware development leading the algorithm development is advantageous or not is left to the reader to decide.

1.2. Mesh Key Frame Interpolation

Mesh key frame interpolation is one of the most traditional starting points of animation. A polygon mesh was used to describe the model and this mesh was manipulated to form new poses, or 'Key Frames.' The idea stems from the days of cell-based animation, where experienced artists would draw the important key frames of a scene and junior animators would fill in the in-between frames. In the same way an animator would construct the important poses or 'frames' for an animation and the computer system would smoothly move from one to the next. The idea of using such a method to implement key frame animation in Cg (Nvidia's high-level shader language) and thus have all the interpolation workload shifted onto the graphics processor is presented by Fernando and Kilguard [4].

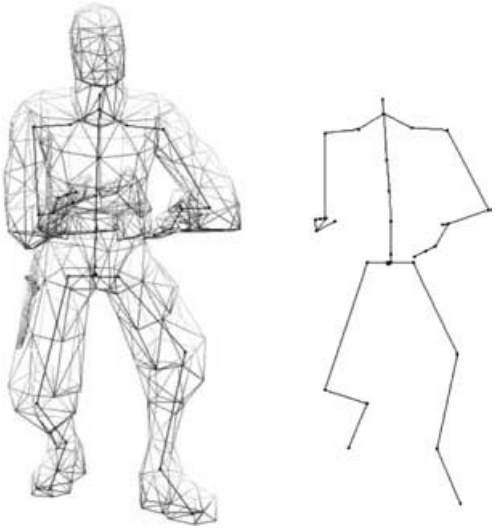

Figure 1: *Example of vertex skinning. (left) Vertex skin and skeleton bones; (right) skeleton bones.*

In general, manipulation of the whole mesh on a per vertex basis is too time-consuming and cumbersome for an animator. Therefore, a way of defining movements at a higher level was needed and, as ever programmers looked to nature. The aim was to separate the work of animating the human from the low level manipulation of the mesh itself. Key framing for each pose is a very inefficient way of storing the data for an animation. In addition in this form the animation is set so that real-time adaptations cannot be easily made (e.g. via inverse kinematics).

1.3. Vertex Skinning

Skinning is one popular technique for animating virtual characters in a scene. With this concept a character is designed as a set of 'bones' over which a skin mesh is applied and attached. The bones are then rotated and translated to give all the posture possibilities for the character. Over this skeleton a 'skin' mesh is applied. In this way it allows realistic animation without having to think about how each vertex in the mesh moves, only the overall changes involved. Each vertex in the skin mesh is attached to one or more bones and, for each bone a weighting is given. Using this matrix of weights the vertex position can be calculated by summing the weighted average of the bone positions. Figure 1 shows an example of

vertex skinning. On the right of the figure is the underlying bone structure upon which the animator works. On the left, the same bone structure is shown with a wire frame representation of the skin.

One of the advantages of vertex skinning is that the skin can be modified irrespective of the bones, as long as the weights are appropriately adjusted. The cost of manipulating the skeleton is a constant overhead; this is insignificant for normal models as they include very simple bone models of the human body. The ability to adjust the skin mesh without having to redefine the animation is highly desirable, making it possible to use level of detail techniques on the skin for example.

1.4. Combining Key Frame and Vertex Skinning

As mentioned Vertex Skinning allows the animator to define a set of bones with which the animation of the character is driven. However, how the manipulation of the bones is defined can take a number of forms. Key frame interpolation can be used to blend between bone positions. Kinematics can also be used to define how the bones are moved, which is an advantage if the animation of the character needs to be affected by external influences. At the present time key frame interpolation is often favored due to its lower runtime overhead, and the ability to be easily hardware accelerated.

The animation of facial expressions is often handled separately from the other character animations. A boneless, mesh only, key frame interpolation can be used and this is often constructed from motion capture. An extension of the bones concept is to use pseudo-bones to model the muscle groups in the human face. An example of this implementation is NVIDIA's 'Dawn' demo [5].

2. Level of Detail

LOD techniques aim to change the geometric complexity of a polygon mesh, while retaining the visual integrity. LOD techniques are often considered for systems where the rendering bottleneck is on the number of polygons that the system can process; the aim is to make more 'effective' use of the number of polygons in the scene. The area of LOD can be split into subsections; the first consists of the opposing strategies of mesh decimation and mesh subdivision techniques. In addition there is the separate area of the error/decisions metrics used to control the LOD strategies.

In the field of virtual humans we mainly consider deformable bodies, not the traditional rigid body work that is used in many of the applications, most commonly architectural walkthroughs and terrain visualization. However, many of the techniques can either be applied to the vertex skin (as view-independent simplification), each part of a connected semi rigid body approximation, or extended to handle non-rigid bodies natively.

2.1. Static Mesh Decimation Techniques

As early as 1976 James Clark considered the benefit of multiple resolutions for models within a scene [6]. The use of hand prepared multiresolution models was a common early implementation of this concept. However, hand preparation is very time-consuming and thus automation techniques were adopted. There are many applications that still use static LOD as the main rendering acceleration method. Often static LOD is used as a preprocessing step for other methods to bring the model to a desired complexity.

An example of a mesh decimation algorithm, which has the ability to join unconnected regions, is given by Garland and Heckbert [7]. The algorithm is offline only, based on contracting pairs of vertices. The main body of the work is upon how the pairs are selected, with each valid pair being ordered based upon the error quadric associated with it. The lowest 'Cost' pair is chosen each iteration and the error values recalculated for the remaining pairs. It is clear that, in its native form, the method places neither emphasis on silhouette preservation nor on any other additional constraints. An additional interest in the paper is that the authors adapt the algorithm to include constraints to preserve boundaries and stop mesh inversion. This philosophy of tailoring algorithms to remove their generality, while improving performance in a specific case lends itself to working with virtual humans where visual importance is not uniform across the mesh. A good starting point on examining static mesh decimation and LOD techniques in general is David Luebke *et al.* [8].

2.2. Dynamic Mesh Decimation Techniques

Progressive meshes were introduced by Hoppe [9–11] and this work formed the basis of many further papers. One of these was Sander *et al.* [12], which showed how it was possible to produce acceptable texture mapping without significant 'Slippage', on a progressive mesh. Dynamic mesh decimation is only useful if the gain from the reduced complexity mesh outweighs the time to compute the decimation. This overhead must be minimized if mesh decimation is to be used for large crowds. The following two papers examine the possibility of using the GPU for mesh decimation.

Shiue *et al.* [13] discuss a possible solution for mesh mutation on future GPUs. In this paper, a framework for a mesh processor that is destined to run solely on a GPU is presented. Modern graphics cards have sufficient memory and efficient vertex processors to yield a significant increase over a CPU only implementation. However, there is one main stumbling block, the current generation of GPUs at the time were the NV30 and R3XX, which did not present the programmer with random memory access which is required for this framework to work. However, in NVIDIA's 6800 (NV40) texture access has been added to vertex shaders, a step toward completely free memory access.

An interesting image quality metric was introduced by Southern in an earlier paper [14]. In this paper, it was shown that decay of the volume of a simplified object is significantly correlated with the decrease in image quality and the quality of the silhouette.

The implementation of their algorithm makes use of vertex programs and this gives them access to the SIMD nature of the current GPUs. The paper presents a single figure comparison for the difference between their software implementation and their hardware optimized solution. 'Our experiments with the non-volatile path ... show a speed up from 71.42 seconds per frame in the case of software interpolation, to between 12 and 14 frames per second on a Athlon 500 MHz processor using an ASUS v8200 Geforce 3.' This is nothing short of a stunning increase, approximately a 800-fold increase in speed due to the use of hardware acceleration. The speed up may well be significantly more pronounced due to their algorithm being heavily optimized for the hardware, at a detriment to the running time of the algorithm in software.

2.3. Mesh Subdivision Techniques

The ability to increase detail in a mesh is also sometimes desirable. The use of static techniques can allow smoothing on a model which does not have the desired complexity level. Dynamic applications allow for scalable levels of detail as a camera approaches a model and also increases tessellation of areas which are to be deformed. Loop's [15] paper is one of the base points for many mesh subdivision techniques. Others include Zorin *et al.* [16] who extended the traditional butterfly method while reducing the number of artifacts on irregular topology. This modified butterfly technique, along with the Loop method and Catmull–Clark subdivision, form the standard against which most subdivision techniques are compared.

Calculating the physical simulation of a model is very computationally expensive, even for a simple model. Kähler *et al.* [17] define a coarse mesh which they use for their physical model, over which they have a dynamically refined geometric mesh. The refinement system they use is based upon Loop subdivision [15]. However, it attempts to do multiple stages of refinement each step. The method starts with the assumption that the mesh is of sufficient quality and thus does not need refining in its undeformed state. On deformation the algorithm checks if any regions are now above a curvature threshold and, if so, it refines them using an edge-splitting method. The curvature function is based on the dot product of the two vertex normals of an edge. The subtriangles are produced efficiently by a lookup table, which stores the 27 different possible combinations of edge splits.

The method they propose would be simple to implement, due to the lack of complex metrics and the fact that the original mesh is not updated. The reason presented for not storing the refinements is that rendering is an insignificant overhead compared to physical simulation. While this may be true in this case, it seems extremely wasteful not to exploit the frame coherency of the mesh.

A hardware-only solution would be advantageous in the sense that the user would be able to use existing methods with no changes to their code. Also when the extra triangles are created in hardware, less data need to be sent over the AGP data bus. Vlachos *et al.* [18] take this approach of implementing local mesh subdivision in hardware. Each triangle is replaced with a cubic Bézier path, which matches the point and normal information of the vertices of the flat triangle. This is the only information used, and thus patches do not join with tangent continuity except at the corners. Cracks are avoided due to each edge being shared between patches and the algorithm is consistent in its output.

The strength of the paper's work is also its greatest failing. The algorithm runs independently of the user's code, and thus control over its results is minimal. The results shown are an improvement over the nonsubdivided models; however, they possess a similar look. The authors call this an 'Organic look'; however, this consistent appearance could be undesirable in the general case.

O'Sullivan *et al.* [19] present the on-going work on a LOD framework not only for geometry but for motion and behavior as well. As part of this framework they used mesh subdivision upon simple starting meshes to give control over the geometric complexity of each Virtual Human. They show comparisons of linear, butterfly, and loop subdivision on a human hand and make use of surface masks to allow the boundaries of the meshes to be maintained, along with creases in the skin. Without these crease masks, all fine detail will be smoothed out. The paper examines many areas of simulating human interaction and is of interest to all trying to simulate a small number of virtual humans interacting in a social environment.

The focus of this paper has been on polygon-based methods; however, it should be noted that tetrahedral-based work has been progressing on similar ideas. For objects that need to be cut and thus have solidity as opposed to being a hollow skin, tetrahedral meshes are more natural. Molino *et al.* [20] present a method for producing efficient tetrahedral meshes.

2.4. Clothing Simulation

An important aspect of visualizing virtual humans is creating realistic clothes upon them. Traditionally, the clothes have been modeled as part of the human mesh. Thus, it has not been possible for the clothes to behave as clothes; they cannot bend or crease as the human moves. It should be noted that realistic cloth modeling is very expensive and gives small visual gain in quality to a user not focusing on the clothes of the model. Ivanov *et al.* [21] suggest the application of online clothes ordering as an application, where it is desirable to see how clothes look upon a virtual human in real-time. Their design

ethos mirrors that of many real-time applications; fidelity is sacrificed in an attempt to dramatically increase rendering speed.

Mass-spring systems are one of the lowest cost methods for modeling the forces on a deforming mesh and are the most commonly used in real-time situations. Ivanov *et al.* [21] also base their implementation on a mass-spring model. However, with added constraints to attempt to overcome some of the drawbacks. They cite that the problem with many current models is that they are 'super-elastic' and the material is allowed to unrealistically stretch large distances. By adapting the velocities associated with the deformations, it is possible to limit the maximum length each spring can reach.

Collisions detection for a cloth draped over a virtual human in real-time is prohibitively expensive in its native form. The method of collision detection is based on image space tests, which reduces the complexity of the collisions detection to a 2.5D problem. The method needs a normal map for each discreet pose, which would be a prohibitively large number of textures for an avatar with a large number of possible animations. This method is independent of the human's complexity and is of linear complexity on the number of vertices in the cloth.

On a SGI (R12000 processor) the implementation presented took over 1 second to 'dress' a human model in a shirt. While this frame rate is not of real-time standards, it is approaching it. Therefore, this method cannot be recommended for a situation other than where cloth dynamics are of the utmost importance. In this small number of cases it may be possible to extend the method using hardware acceleration to get almost real-time results.

Daubert *et al.* [22] concentrate on attempting to visually replicate the complexity of woven garments in real-time. Due to a limited amount of geometric primitives that can be allocated to the material in a real-time system, modeling each weave separately in geometry would be unfeasible. Texture mapping can replicate small details, however, at closer inspection the true flatness of the surface is apparent. While bump mapping and now normal map techniques have come some way to correct this problem, there is still a lack of occlusion and self-shadowing effects required for a more realistic appearance.

In the paper, Daubert *et al.* [23] model a single stitch of the repeating pattern and from this generate normal and lighting maps. With this information IBR methods can be used to replicate and relight the image. Data acquisition is a significant problem, as with all systems that use BRDF's to model the physical properties of a material. This system takes 15–45 minutes to construct the required information from a 3D model. Due to the repeating nature of the material they are modeling, aliasing problems are significant and thus they employ a MIP-Mapping-based solution.

The results of this work are still too slow for real-time (1 fps), however, with further work on incorporating the use of modern graphics hardware the authors believe it is possible to improve this to truly real-time rates. Again, unless the clothes modeling is extremely important to the application, even with significant improvements it consumes an unfeasible amount of processing time.

Volkov and Ling [24] present the idea of adaptive LOD for physical simulation of cloth meshes. Areas where the curvature is beyond a threshold are refined and those that fall below are simplified. In this way the flat areas of the cloth that would otherwise be overly tessellated for a simulation, can be modeled simplistically, leaving the areas of folding to be concentrated on. The aim of this work is for a higher performance cloth simulation, and while their results are not real-time they are a significant improvement over cloth simulation without such techniques.

It is still some time away before we can have truly believable simulations of the clothes upon a virtual human being calculated in real time. Baraff *et al.* [25] have done much work in the non-real-time field and have shown how important correct clothing is to stop humans looking rigid and plastic. A clothing simulation for an individual is still not at real-time, clothing simulation for a whole crowd will continue to be outside the power of computer systems for the near future.

2.5. LOD Selection

Southern *et al.* [26] presents a paper that concentrates on the implementation of LOD tailored toward exploiting the SIMD architecture of modern programmable graphics hardware. Within this paper they focus on the factors which govern a seamless change. They propose these to be:

- *Visual continuity.* The eye should not able to detect any discrete changes in the model.
- *Geometric continuity.* Only at the position of an existing vertex can another vertex be added or removed.
- *Frame-rate continuity.* A consistent frame refresh rate is desirable, which implies each frame takes a constant amount of time to render irrespective of the geometric complexity within the view frustum.

As mentioned in the paper, visual continuity and frame-rate continuity are often impossible to be satisfied in a general environment at the same time. They call the times where both cannot be maintained as a 'volatile scene', where large amounts of the scene are being changed between frames. Reference is made to Funkhouser and Sequin [27] which states that it is only by prediction that a volatile scene can be handled without breaking frame-rate continuity or visual continuity. Hidalgo [28] is an example of method which implements a motion prediction method combined with other speed-up techniques with the aim of keep a constant frame rate. The

Figure 2: *Implementation of a crowd with static impostors.*

system uses a Kalman filter to predict position and orientation of the camera. While motion prediction of the viewing position in the field of virtual humans could be useful, with a dynamic scene there would be several extra considerations that may not make it as suitable solution. It would require a crowd behavior system which was deterministic, for at least the near future.

3. Visual Impostors

Image-based rendering (IBR) systems have been subject to a large amount of research. Visual impostors introduce image-based techniques into a geometric rendered environment. The aim is to replace sections of the scene with an image representation textured onto a simple, normally a quad, geometric shape. In this way the rendering time of an object is reduced to a constant texturing overhead. Figure 2 shows an implementation of static impostors, with the white outline indicating the edge of the billboard.

3.1. Static Impostors

With static impostors the textures are generated offline, stored and merely loaded at runtime. Each texture is normally valid for a limited group of viewpoints. Through the use of certain image manipulation techniques, e.g. image warping and storing extra information, e.g. multi-depth images (MDI), it is possible to extend the set of viewpoints in the image it is valid for. Tecchia and Chrysanthou [29] cover, in addition to the problems of graphically rendering a large crowd, handling the collision and behaviour of the virtual humans in real-time. Both of these topics are outside the scope of this paper, but are of interest to anyone trying to construct a simple simulation on which to test the techniques.

The graphical techniques implemented by Tecchia and Chrysanthou [29] use no image warping or blending of the static impostors. This allows for a larger amount of the CPU power to be dedicated to rendering the scene. The direct side-effect of this is that it reduces the viewing area for which an impostor is valid. In an attempt to counteract this the method does not always choose an impostor plane perpendicular to the view point. The concept behind this is to choose a plain that minimizes the visual error between two different pregenerated views. They define a method for choosing this plane, which minimizes 'the sum of the distances of the sampled points, and the projection plane given a camera position from where the sample image is created'. Through this technique the number of impostors needed to be stored is reduced while keeping the same visual quality. Dynamic impostor systems could also make use of this work and thus increase the time of validity of each impostor. However, it would only be beneficial if an implementation of this technique took less time to calculate than the saving in time over the fewer refreshes of the impostor in question.

To reduce the storage overhead of the textures the property that humans are asymmetric is exploited, along with the use of the OpenGL texture compression extension. Even so, storing a different texture for each human in a scene becomes impractical as the number of humans in the scene increases. Changing the shape of the texture at runtime to resemble different human forms is a very complex task. For simplicity and speed only a technique which gives color variety to the character's skin, clothes, and hair is used. With the alpha channel of their texture they can divide up the character into different sections and color each one separately using multipass rendering.

Static impostors alone, as used by Tecchia and Chrysanthou [29], lead to a very limited crowd scene. As mentioned, the texture overheads for static impostors are significant for a single animation and a single model. Only a small number of possible animations could hope to be represented with impostors without a crippling number of textures. Also, alaising effects become significant with static impostors as they approach the camera if they are not of a sufficient size. Again a compromise between impostor size and memory consumption has to be made.

With static impostors alone it is hard to allow for external factors, such as changes in lighting within the scene. There has been some work by Aubel *et al.* [1] and Tecchia and Chrysanthou [29], where normal maps are used to implement view-dependant lighting. Through the use of per-pixel shading via fragment shaders, normal maps can be efficiently implemented on modern graphics processors. Loscos *et al.* [30] discuss many of the problems with handling shadows on impostors (see Section 4).

Billboard clouds are an extension of the static impostor concept. 3D models are simplified onto a set of planes, each plane with texture and transparency maps. In Decoret *et al.* [31] make the claim 'for extreme simplification our approach combines the strengths of mesh decimation and image-based impostors.' In the paper a weakness of LOD methods is identified, which is their inability to satisfactorily represent

extreme simplification representation of complex, disconnected objects. Impostors can combine multiple objects well but suffer from limited parallax; while billboards can fuse multiple objects without sacrificing parallax effects. While the method is designed only for static models, it may be possible to extend it into handling limited dynamic objects, e.g. a walking human at a distance. Careful memory management would be needed, due to the need to store 30–100 textures for each model.

3.2. Dynamic Impostors

Instead of converting the geometry to a texture offline, the impostors can be generated for the current view. Aubel *et al.* [1] present a completely IBR method for virtual humans. They use a skeleton and skin method to represent a virtual human. If the posture, defined by the bones, varies from the current pose by more than the error threshold then the image is updated. In this way only a small number of key frames of the animation are shown to the viewer, relying on the viewer's brain to fill in the missing frames. This method approximates many animations successfully; however, any small subtleties in the animation are lost. For example, breathing or other small movements would fall below the error threshold and therefore be lost.

Aubel *et al.* [1] also differ from many methods by not using a single quadrilateral plane to represent a virtual human. One reason given is that a single plane quadrilateral can cause visibility issues. Instead the virtual human is divided up into 'coherent parts', where each part is a section which cannot overlap itself. They provisionally generate 17 sections for a virtual human. One advantage of this technique is that if only a part of the character is being animated, for example only one arm, then a subsection of the impostor is all that needs updating.

In a later paper, Aubel *et al.* [32], a LOD for the skins of a virtual human is described. This LOD structure is for the skin mesh only, not bones and thus it cannot do LOD of animation. They present a body representation that has the ability to generate skin deformations due to muscle contractions, a feature rarely seen. It is from these simplified geometric meshes that the impostors are generated. The smaller the projected size of the impostor onto the view plane, the smaller the generated texture, which avoids aliasing effects of scaling incorrectly sized impostors. Like many impostor techniques, trying to maintain a constant frame rate is a challenge. The updates are prioritized; only allowing a certain amount of geometry to be rendered and, thus impostors, each frame. This idea is based on the belief that the human visual system is more willing at times of rapid motion to accept smooth movement at the expense of lower image quality.

Due to the fact that the impostor is being generated in the current scene and in the current lighting model, the impostor is able to display these attributes. An approximation of view-dependent lighting is possible; however, this can be visually distracting if the impostor is not updated regularly enough. While techniques such as image warping methods for textures can be used to extend the life of an impostor, they can not handle adjusting the view-dependant part of the lighting. Through the use of normal maps, it is possible to add lighting to impostors. However, at the current time, there is no efficient way of constructing and using normal maps solely on the graphics card without having to move them to main memory for format conversion.

4. Lighting and Shadowing

The human visual system makes heavy use of lighting and shading to interpret the world around us. Photo-realistic lighting is very complex to simulate and at the present time it is far beyond the real-time capabilities of current high end computer systems. Until the computational power grows to the extent a real-time full radiosity solutions is possible, techniques are needed to bridge the gap between what is desired and what is possible.

4.1. Improved Lighting

Self-shadowing and self-inter-reflection play a subtle but a significant part in the lighting of human faces. Without it facial expressions are harder to see and so the virtual humans look false and lifeless. Radiance self-transfer is costly to compute at runtime and thus it is advantageous to be able to handle parts of it offline. Sloan *et al.* [33] use the idea of attempting to parameterize the lighting equations, moving some of the calculations offline and constructing lighting matrices which can be used at runtime. The work in this paper and in Kautz *et al.* [34] are very closely linked. They describe a method based on spherical harmonics and with this the light integral at a point can be limited to 25 calculations. While this is still a significant number, it is possible for use with real-time application due to the power of modern graphics cards, if it is used sparingly. It should be noted that to gain truly interactive frame-rates either a fixed light or a fixed viewpoint is need. The results presented do not make full use of modern hardware acceleration, as the features were not available at the time of the paper. With this addition it may be possible to get interactive frame rates without the heavy constrains of a fixed point of view or a fixed light source.

In Bastos *et al.* [35] discussion is made on how modeling the view-dependant elements of lighting in an architectural walkthrough can greatly improved the photo-realism. The aim is to introduce view-dependant lighting into a hybrid geometry and IBR system. While architectural walkthroughs can take advantage of the static nature of the scene, it provides ideas which can be adapted for virtual humans. Through their work of decomposing the lighting down into different sections, it is possible to obtain glossy reflections in real-time. One of the most attractive parts of their method is it runs in constant time per reflector. This is ideal in a real-time

environment to avoid frame rate 'spikes' which can severely detract from the feeling of interactivity of a scene. There is, however, significant storage overheads associated with this method.

While glossy reflections do not play a large part in the lighting of skin itself, there are certain circumstances where is does play a larger part — when the skin is wet, e.g. rain or sweat. In addition human hair and the synthetic fibers that make up certain types of clothes, have significant glossy reflection elements. Therefore, while such reflections do not feature heavily in the displaying of humans at the current time, they do form a vital part of the lighting model needed for a realistic virtual human.

4.2. Shadows

Computer displays are flat, two-dimensional projections of a three-dimensional world. Without lighting or movement any shape looks as flat and two-dimensional as it truly is. It is from motion and lighting that the human eye infers three-dimensionality from the 2D projections we are presented. However, the concept of shadows, while featuring highly in the real world, has been either ignored or loosely approximated up until recently.

4.3. Shadow Volumes and Shadow Maps

Shadow maps, introduced by Williams [36] in the 1970s, are an adaptation of the depth map idea. The main problem with shadow maps is that they are texture-based solutions and thus suffer aliasing when a shadow map's element corresponds to more than one screen pixel. Much research has been done to combat the aliasing problems of shadow maps. However, shadows maps are far less expensive to calculate than the other main shadow technique, shadow volumes.

Shadow volumes avoid the problems of aliasing by being geometry based. First introduced by Crow [37], a shadow volume is a semi-infinite frustum extending back from the edge of a polygon away from the light. Through the use of a stencil buffer (available on most modern GPUs) shadow volumes can be implemented in hardware. The shadow quality is high with shadow volumes, though with many large shadowed areas in a scene it can soon consume the entire fill rate of most modern systems. Thus shadow volumes are suited to objects which need high-quality shadows and do not cast large complex shadow volumes.

Fast calculation of shadow volumes utilizing hardware acceleration has produced several proposed implementations. McGuire *et al.* [38] is one paper that is of interest to all working in the field. In addition to the concepts of the algorithm, the fundamentals of how to construct an implementation is given in detail. The techniques' aim is to reduce the amount of rasterization traditionally involved with shadow volumes, at the expense of increased vertex processing. This is due to the fact that in the current architectures graphics algorithms are fill rate bound as opposed to polygon bound. The algorithm is split into two passes. The first pass, which they call the 'shadow determination pass,' separates the screen-space into regions of light and dark. This is the section that the work focuses on optimizing. The second section is the 'illumination pass' which is used to compute the lighting for the lit areas.

A major drawback of this method is that identical vertex arrays are stored in the main system memory and the video card memory. This has significant impact on the use of hardware vertex skinning, as the same operations have to be replicated in software on the vertex array in main memory. The reason for needing to do this is so that the silhouette detection test can be run on the deformed pose, not the static starting pose.

Brabec and Seidel [39] claim to have a system that solves the main problem which McGuire *et al.* [38] suffer from. The main interest of the paper is their method for silhouette detection that is implemented solely on the graphics processor. Potentially fast shadows for virtual humans could be possible with only minimal changes to the main program code, through the use of a library shader with the method implemented within it. The vertices, in world co-ordinate space, are stored in a floating point texture. Each vertex is assigned an index, which indicates where in the texture it is stored. Once this texture has been constructed a silhouette detection test can be done upon it.

The implementation of the algorithm highlights that without advances in the current features of graphics hardware the true potential is not realized. Texture access is not allowed at the vertex shader level, where it would be far more natural to process the vertex lists stored in the texture than at the pixel level. Due to these restrictions, it is necessary to move data from the two buffers, to main memory and immediately move it back to the graphics card in the form of vertex attribute data. Therefore, the paper proposes interesting ideas, but it does not truly solve the problem.

A part solution to the problem of drawing shadows for animated crowds being rendered using visual impostors is presented by Loscos *et al.* [30] (based on the work by Aubel *et al.* [1]). Two different elements of shadow computation are considered, static geometry onto dynamic characters and dynamic characters casting shadows onto a flat ground surface. The shadows produced are for a single, fixed light source and are only sharp shadows. In this way, the solution can be considered a work in progress, as they do not cover among others the problems of dynamic characters casting shadows onto each other. Figure 3 is an example implementation extended to multiple light sources.

The framework used for the rendering of crowds in the virtual city makes a restriction on the scene being 2.5D. Height maps are used to describe changes in the level of the scene. The camera is limited to eye level and it is not possible to

Figure 3: *Implementation of shadow maps for static impostors.*

represent objects such as bridges and building overhangs. For each building a shadow height map is constructed offline and processed at runtime. If a character is within a shadow map, the stored shadow height value is compared to the height of the character. If the character is considered partially covered, then a second texture is used to show the character's legs to be in shadow over the top of the impostor texture. Similarly if the character is decided to be in full shadow, then a texture where the whole character is in shadow is applied over the top of the impostor texture. This is a fast but crude way of doing shadows for impostors caused by static geometry (e.g. buildings). The aim of their work is not simulation, but approximation of shadows. The results are a significant improvement to current shadowless crowd visualization techniques.

The casting of shadows from the characters to the ground is based on the re-projection of the impostor onto the ground, from the light source's position. Again a full shadow texture is used instead of the original color of the textures, thus giving the impression of a cast shadow. The texture of the character is already in memory, and therefore there are no extra stored requirements for this shadow method. However, this method only works onto a horizontal floor, it cannot cast onto other objects be they static or dynamic. The result of their work is the ability to handle shadows for approximately 10,000 crowd entities in real-time.

4.4. Hybrid and Adapted Shadow Methods

Shadow Silhouette maps, an extension to shadow maps presented by Sen *et al.* [40], yield higher-quality shadow boundaries than traditional shadow map methods. The method generates a depth map and a silhouette map in the first pass for rendering from the view of the light. In the second pass the scene is rendered from the viewer's perspective. They present the idea that the way to combat the problems with the shadow boundaries is through the use of a silhouette map, which stores extra information about the boundaries of the shadow.

A silhouette map is defined as a 'texture whose texels represent the (x,y) coordinates of points that lie on the silhouettes of objects' [40]. This is not a native texture format supported by hardware acceleration, and thus a significant performance penalty is incurred. The results of the implementation they present are a marked improvement on the visual quality of the edge boundaries of the shadows. An interesting point of the work is that shadow silhouette maps use significantly less bandwidth than shadow volumes, but the implementation showed no performance increase over shadow volumes. This is explained by the fact that current graphics cards are optimized to handle shadow volumes efficiently via the stencil buffer. In its current form it is hard to justify the use of silhouette maps as shadow volumes produce better quality shadows faster.

Govindaraju *et al.* [41] combine shadow mapping and shadow volumes in a hybrid method. The goal is to achieve interactive shadows, where the light source is not fixed, on very large models. The hybrid method they present has an additional aim to significantly reduce the aliasing effects associated with shadow maps without incurring the heavy fill-rate costs of only using shadow volumes. The main focus of the work is splitting the rendering of the scene into different sections, which can be run in parallel to each other.

Improved techniques for computing the potentially visible set (PVS) are presented, through the use of hierarchical representation, LODs, and image-space occlusion queries. PVS are computed for the eye and the light, and a cross-culling method between the two PVSs is introduced. With this system, models can be handled without topology or connectivity assumptions. The implementation they present is on three dual processor 1.8 GHz Pentium 4-based machines, each with a NVIDIA Geforce 4. This allows them significantly more processing power than most algorithm implementations considered in this paper, and as such the results should be considered appropriately.

There are some strong features of this method, the light source is not fixed and the shadows show little aliasing even on long thin objects. It seems that it would be possible to render a large crowd with high-quality shadows. The main limitation of this method is that for each light introduced into the scene, another graphics processor (and so another machine) is required. In addition to the extra processors it is not clear how large an impact on the rendering speed the extra cross-culling and shadow computations would have.

Hasenfratz *et al.* [23] present an in-depth review of soft shadow methods in real-time. The growth in this area is being driven by the development in power and features of the modern graphics cards. The survey gives a good overview of

the subject and is an excellent introduction to soft shadows in real-time applications.

5. Supplementary Techniques

Modern Shader Languages allow, through a high-level language, control over the graphics processor units (GPU). The ability to control the way objects are shaded has existed in offline rendering for a long time. The aim is that designers can produce custom shading methods, specifically tailored for the situation and thus get higher-quality results. The second main purpose of high-level shader languages is to expose programmers to the hardware accelerated commands available. In this way programmers can write shaders designed to give realistic shading of skin, hair, or clothes with customizations to the way lighting is calculated for each in a way they would not be able to otherwise. In addition to this they can implement features beyond just lighting, for example vertex skinning or mesh mutation that take advantage of the hardware accelerated vector processing.

The human visual system places a strong emphasis on the human face, thus it should be given higher priority over other elements of the representation. Paris *et al.* [42] propose a reduced fidelity model for relighting facial images with the aim of providing a real-time solution. Through the use of parametric modeling of the skin, combined with textures to add 'roughness,' they propose to relight captured images as though they were part of the current scene.

Capturing the input image is nontrivial, as any parts of the image that are over-or underexposed will cause loss of skin detail for that section. Once the image has been captured, all shading must be removed from the input images as a preprocessing stage. The relighting of the picture algorithm is essentially a Phong shading system with addition detail being provided from a texture. The eyes and mouth have to be handled separately due to their problematic nature. Specular highlights are added to the eyes by modeling them as hemispheres. While this helps, the results for both eye and mouth can give false visual clues to the viewer over where the light is coming from due to shadow artifacts remaining from the original images.

The resulting images, while a significant step above traditional polygon-based solutions, still do not look 'natural' to the human eye. The speed of this method, using a Geforce 4 Ti 4400, is 60 Hz for multiple light sources for a complex mesh (12,250 vertices). Thus, it is possible to incorporate this method into a system for rendering virtual humans in real-time. Their system does not exploit Vertex or Pixel shaders and thus could be efficiently run on a wide range of systems. One point made by the authors is that the inclusion of shadows into the system, via shadow maps, causes a significant performance hit and thus should be used sparingly.

Olano *et al.* [43] present the idea of extending LOD work to include Shader LOD. The idea that objects can be drawn with different complexity shaders is proposed, much in the same way as objects are drawn with different geometrically complex meshes. They show that with this scalable architecture the objects in the foreground can be rendered with higher-quality shaders, at the detriment to less important objects in the scene. The main aim of the work presented in this paper is to reduce the number of textures used per object for the lighting and shading calculations. Modern techniques of texturing objects often involve multiple textures to create the desired effect, using the extra textures to store precomputed expressions. Two separate simplification methods are discussed, firstly lossy simplification which replaces texturing with an approximation. A least-squares error metric was used to control this simplification. However, in the general case, controlling the quality of the output is indicated to be problematic. Lossless simplification is the other method, which works by reducing the number of textures used by combining textures into a single texture.

While the number of textures access per object is reduced, there is a significant increase in the number of textures needing to be stored. Graphics memory on many modern cards is plentiful but when there is use of multiple texture intensive techniques, texture paging is needed. If the textures are being swapped in and out of graphics memory too often this technique will be outweighed by the time taking to move the texture onto the graphics card memory.

Virtual human models are created in a number of different ways. Many methods do not produce high enough quality meshes and therefore it would be advantageous to preprocess all meshes for a program. An example of this is in progressive mesh and mesh refinement techniques, where a regular connectivity is often a required for the algorithm to give satisfactory results in real-time. A remeshing algorithm is introduced by Surazhsky *et al.* [13] which is robust and does not have unreasonable time requirements (approximately 10–20 seconds for a under 10,000 polygon model). The method is based on local operations on the mesh, thus avoiding the expensive global parameterization of many previous techniques. The techniques are designed for triangle quality, and while the results are a good approximation of the original mesh, errors are introduced. In the rendering of virtual humans in real-time this error is likely to be insignificant to the improvement of the quality of the mesh for LOD work.

6. Conclusion

LOD methods still suffer from one major drawback: at extreme levels of simplification the model's visual fidelity is very poor. This makes them unsuitable for rendering crowds of humans in real-time. However, when used in hybrid implementations, with IBR, LOD techniques show good promise for rending virtual humans in the foreground. LOD implementations solely on GPUs are almost possible and thus the penalty for adjusting the mesh at real-time should be significantly reduced. One of the major remaining problems is frame

rate spikes, which are common to LOD and visual impostor work. At the present time most techniques are re-active not pro-active in their handling of the viewer's movement. The incorporation of motion prediction systems into these methods should have significant benefits for frame rate consistency.

Current visual impostor implementations look very promising for crowd simulation. Their ability for extreme polygon reduction (unlike LOD) means an entire crowd can be modeled without overwhelming the polygon capabilities of a system. This paper identifies three main areas for future research, first removing the 2.5D limitation common to the majority of the methods without a prohibitive performance hit. Second, visual variety in the crowd is currently extremely limited. This is due to texture memory constraints, a different texture cannot be used for each entity in the crowd. Therefore, improved ways to give pseudo-variety are needed for texture-based solutions. The memory overheads associated with IBR for virtual humans need to be formalized, as other parts of the scene may also use texture intensive techniques, introducing the need for texture paging.

Finally, the casting of shadows by the impostors has only been attempted partially for a crowd. Casting shadows onto nonflat static geometry and onto other dynamic elements still needs to be addressed. While many solutions exist for these in other areas, due to the number of shadows needing to be drawn, a lower cost (possibly lower accuracy) method is needed. Normal maps allow for dynamic relighting of impostors; however, these can not be constructed and used at runtime without format conversion, which makes them unfeasible for dynamic impostor implementations. Lighting and shading in real-time is an area where much research is being undertaken. Virtual humans cannot take advantage of the optimizations that are used for static objects. Parameterizing the lighting equations looks a promising technique, though implementations for a deformable object do not run in real-time. Accurate hard shadow techniques are possible in real-time for a reasonable number of virtual humans, however a more realistic soft shadow lighting model for a crowd of 10,000 humans is still unachievable on a consumer-level hardware.

In overview, visual realism is the main obstacle to be overcome in the current real-time virtual human environments. LOD for modeling small numbers of high detail deformable models and IBR for entire crowds are tried and tested methods which are continuing to evolve. We are on the verge of a significant improvement in visual quality, once the next generation of GPUs on which the methods can be solely run are released. Poor lighting models continue to plague a number of the current implementations and this is an area where significant work is still needed for real-time virtual humans.

References

1. A. Aubel, R. Boulic and D. Thalmann. Lowering the cost of virtual human rendering with structured animated impostors. In *WSCG,* Plzen, Czech Republic, 1999.

2. D. Cohen-Or, Y. Chrysanthou, C. Silva and F. Durand. A survey of visibility for walkthrough applications. *IEE Transaction on Visualisation and Computer Graphics*, 2003.

3. T. DeRose, M. Kass and T. Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics* (1998), ACM Press, pp. 85–94.

4. R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*, Addison-Wesley Longman Publishing Co., Inc., 2003.

5. NVIDIA. Available at http://www.nvidia.com/object/demodawn.html.

6. J. H. Clark. Hierarchical geometric models for visible-surface algorithms. *SIGGRAPH Computer Graphics*, 10(2): pp. 547–554, 1976.

7. M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics 31*, Annual Conference Series pp. 209–216, 1997.

8. D. Luebke, B. Watson, J. D. Cohen, M. Reddy and A. Varshney. *Level of Detail for 3D Graphics*. Elsevier Science, Inc., 2002.

9. H. Hoppe. Progressive meshes. *Computer Graphics*, 30: 99–108, 1996.

10. H. Hoppe. View-dependent refinement of progressive meshes. *Computer Graphics*, 31: 189–198, 1997.

11. H. Hoppe. Efficient implementation of progressive meshes. *Computers and Graphics* 22(1): 27–36, 1998.

12. P. V. Sander, J. Snyder, S. J. Gortler and H. Hoppe. Texture mapping progressive meshes. In *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press/ACM SIGGRAPH, pp. 409–416, 2001.

13. L.-J. Shiue, V. Goel and J. Peters. Mesh mutation in programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, Eurographics Association, pp. 15–24, 2003.

14. R. Southern, P. Marais and E. Blake. Generic memoryless polygonal simplification. In *Proceedings of the First International Conference on Computer Graphics, Virtual Reality and Visualisation*, ACM Press, pp. 7–15, 2001.

15. C. Loop. *Smooth Subdivision Surfaces Based on Triangles*. Master's thesis, University of Utah, 1987.

16. D. Zorin, P. Schröder and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. *Computer Graphics 30*, Annual Conference Series, 189–192, 1996.

17. K. Kähler, J. Haber and H.-P. Seidel. Dynamic refinement of deformable triangle meshes for rendering. In *Proceedings of the 19th Annual Conference on Computer graphics*, pp. 285–290, 2001.

18. A. Vlachos, J. Peters, C. Boyd and J. L. Mitchell. Curved pn triangles. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, ACM Press, pp. 159–166, 2001.

19. C. O' Sullivan, J. Cassell, H. Vilhjalmsson, J. Dingliana, S. Dobbyn, B. MacNamee, C. Peters and T. Giang. Levels of detail for crowds and groups. In *Computer Graphics Forum*, vol. 21, pp. 733–741, 2002.

20. N. Molino, R. Bridson and R. Fedkiw. Tetrahedral mesh generation for deformable bodies. Available at http://graphics.stanford.edu/papers/meshing-sig03/.

21. T. Vassilev, B. Spanlang and Y. Chrysanthou. Efficient cloth model and collisions detection for dressing virtual people. In *ACM/EG Games Technology Conference*, January 2001.

22. K. Daubert, H. P. A. Lensch, W. Heidrich and H.-P. Seidel. Efficient cloth modeling and rendering. In *Rendering Techniques 2001: Proceedings of the 12th Eurographics Workshop on Rendering*, 25–27 June 2001. pp. 63–70.

23. J.-M. Hasenfratz, M. Lapierre, N. Holzschuch and F. Sillion. A survey of real-time soft shadows algorithms. InP. Brunet and R. Scopigno, *Eurographics*, State-of-the-Art Report. *Computer Graphics Forum* (2003), no. 22(4).

24. V. Volkov and L. Li. Adaptive local refinement and simplification of cloth meshes. *ICITA*, November 2002.

25. D. Baraff, A. Witkin and M. Kass. Untangling cloth. *ACM Transactions on Graphics*, 22(3): 862–870, 2003.

26. R. Southern and J. Gain. Creation and control of real-time continuous level of detail on programmable graphics hardware. *Computer Graphics Forum* 22(1): 35–48, 2003.

27. T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th Annual Conference on Computer graphics*, ACM Press, pp. 247–254, 1993.

28. E. Hidalgo. *Hybrid Geometric - Image Based Rendering for Virtual Reality*. Ph.D. thesis, 2003.

29. F. Tecchia and C. L. Y. Chrysanthou. Visualizing crowds in real-time. *Computer Graphics Forum* 21(4): 753–765, November 2002.

30. C. Loscos, F. Tecchia and Y. Chrysanthou. Real-time shadows for animated crowds in virtual cities. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM Press, pp. 85–92, 2001.

31. X. Decoret, F. Durand and F. X. Sillion. Billboard clouds. In *Proceedings of the 19th Conference on Computational Geometry*, ACM Press, pp. 376–376, 2003.

32. A. Aubel, R. Boulic and D. Thalmann. Real-time display of virtual humans: Levels of detail and impostors. *IEEE Transactions on Circuits and Systems for Video Technology*, 2000.

33. P.-P. Sloan, J. Kautz and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, pp. 527–536, 2002.

34. J. Kautz, P.-P. Sloan and J. Snyder. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *Proceedings of the 13th Eurographics Workshop on Rendering*, Eurographics Association, pp. 291–296, 2002.

35. R. Bastos, K. Hoff, W. Wynn and A. Lastra. Increased photorealism for interactive architectural walkthroughs. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, pp. 183–190, 1999.

36. L. Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Computer Graphics* 12(3): 270–274, 1978.

37. F. C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH Computer Graphics* 11(2): 242–248, 1977.

38. M. McGuire, J. F. Hughes and K. T. Egan. Fast practical and robust shadows. Available at http://developer.nvidia.com/object/fastshadowvolumes.html.

39. S. Brabec and H.-P. Seidel. Shadow volumes on programmable graphics hardware. *Computer Graphics Forum*, 22(3): 433–440, September 2003.

40. P. Sen, M. Cammarano and P. Hanrahan. Shadow silhouette maps. *ACM Transactions on Graphics* 22(3): 521–526, 2003.

41. N. K. Govindaraju, B. Lloyd, S.-E. Yoon, A. Sud and D. Manocha. Interactive shadow generation in complex environments. *ACM Transactions on Graphics* 22(3): 501–510, 2003.

42. S. Paris, F. Sillion and L. Quan. Lightweight face relighting. In *Proceedings of Pacific Graphics*, October 2003.

43. M. Olano, B. Kuehne and M. Simmons. Automatic shader level of detail. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, Eurographics Association, pp. 7–14, 2003.