Shadow Mapping and Visual Masking in Level of Detail Systems [1]

Ronan Dowling

May 14, 2003

[1]Submitted under the faculty supervision of Professor Gary Meyer, in partial fulfillment of the requirements for the Bachelor of Arts degree, *summa cum laude*, Department of Computer Science, College of Liberal Arts, University of Minnesota, Spring 2003.

Abstract

This paper examines the effects of real-time lighting and shadows on the perception of low detail three-dimensional models. It does so through the creation of a simulated slide projector that can be used to study the masking effect of textures in a level of detail system. Surface textures have long been known to have an effect on a viewer's perception of model detail, but since surface textures are generally static, they are of little use in dynamic LOD systems. This paper looks at how a dynamically changing projected texture, in the form of a simulated slide projector with realistic real-time shadow generation, affects model detail perception. The results of this experiment show that light and shadow falling on a model can have a dramatic effect on the perceived quality of models, which could allow such factors to be taken into account when developing a sophisticated level of detail system.

1 Introduction

Shadows are a vital part of the way humans perceive images. Without realistic shadows, a computer generated image will look flat and unconvincing. Unfortunately, shadow generation is traditionally a computationally expensive part of image rendering and is, therefore, often omited from real-time graphic engines. There are, however, some shadow generation techniques that are efficient enough to work at interactive rates on today's graphic hardware. Unfortunately these algorithms can require multiple rendering passes for each frame, and are best best suited for scenes consisting of relatively few polygons.

Level of detail (LOD) systems could be a valuable tool in enabling the use of shadows in real-time graphics. LODs lower the rendering time of a scene by reducing the number of polygons needed to draw a given mesh. They simplify models while maintaining image fidelity by deleting those polygons with a visual impact that is negligible in the given context. A key element of LOD is the method used to simplify, or decimate a mesh in order to reduce its level of detail. Traditional mesh simplification schemes rely on geometric comparisons between the original mesh and the decimated one in order to judge quality and guide simplification. Turk and Lindstrom[7] challenged this approach when they introduced image-based simplification which compares images taken of the models rather than the mesh geometry. Image based comparisons allow simplification algorithms to take advantage of perceptual phenomena such as texture masking and lighting effects.

Image-based analysis is a new development in the field of LOD, but it is an important one. Though most LOD research judges the quality of a simplified mesh based on its geometric similarity to the original, a user of an interactive application will be more concerned with the visual fidelity of the object. In simple scenes, geometric similarity and visual similarity are roughly equatable, but modern graphic contexts usually contain scenes that are anything but simple. Cutting-edge interactive graphics employ advanced lighting, surface textures and other special effects, all of which can have a significant impact on a viewer's ability to perceive error in low-detail models. The next generation of LOD systems will need to take both visual and geometric factors into consideration if they are to fully optimize meshes for modern real-time applications. In turn, sophisticated, image-based LODs could help make feasible some of the more expensive visual effects, such as shadow generation, at interactive frame-rates.

Figure 1: The results of placing two models of different complexities in front of a simulated slide projector.

This paper describes the creation of a simulated slide projector with real-time shadow generation. The motivation for this project comes from the need to develop a tool to help demonstrate the results of a new image-based mesh simplification scheme. This new scheme uses lookup tables to allow the use of a more sophisticated image comparison method to judge the quality of a simplification. This image comparison method takes into account some of the subtler aspects of human visual perception, including that of visual masking.

It has long been known that a pattern or texture on the surface of a mesh can affect a viewer's ability to discern error in a polygonal mesh. This fact allows for the reduction of a mesh based on it's surface characteristics, but has not, thus far, been used in the development of a LOD system that can dynamically reduce the number of polygons in a mesh. One reason for this is that, although many real-world objects have interesting surface textures, these textures are generally static and will not change during the course of an animation or simulation.

A simulated slide projector provides a natural way to alter the texture of an object dynamically. While it is unrealistic and unintuitive to alter the surface properties of an object during an animation, it is entirely natural to expect a pattern of light projected onto a model to change over time, and thus affect the way that model is perceived. Shadows help to add another level of realism to the simulation, and can themselves be a natural way in which the pattern on the surface of an object changes. Figure 1 shows how a pattern of light cast onto an object can affect perception of error in that object. The top row of the figure shows meshes of two different complexities under a simple spotlight. The difference in quality of the models is quite apparent in this context, but is almost imperceptible when the meshes are each placed under a more complicated pattern of projected light.

2 About LOD Systems

LOD systems attempt to decrease the rendering time for a scene by reducing the detail of polygonal meshes. LOD is mainly used in real-time graphic applications, such as video games and interactive visualization programs, to allow better frame rates and more complex scenes.

Polygonal meshes are an integral part of many real-time graphic engines. Characters in computer games, furniture in architectural models, and scien-

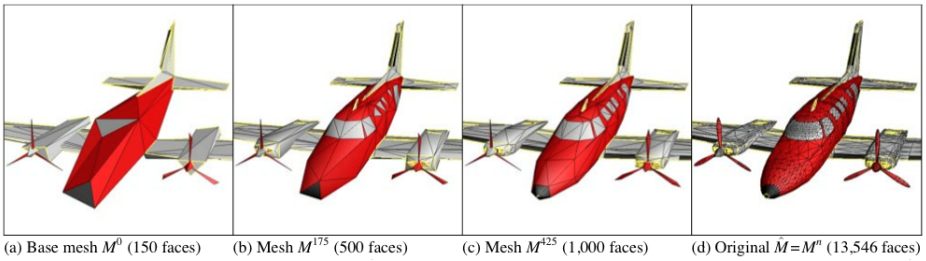

Figure 2: Multiple levels of detail of a single model. From Hoppe[4].

tific visualization data can all be represented using models that consist of a network, or *mesh* of simple polygons, usually triangles. A mesh that contains more polygons will have a more refined shape and better visual fidelity but will take longer to render and therefore reduce the frame-rate of an interactive application. A mesh with a low polygon count will take less time to render but will contain much less detail.

LOD attempts to improve the performance of real-time graphic software by swapping complex meshes for simpler ones without significantly reducing the quality of the rendered image. Image fidelity is maintained by taking advantage of a number of different aspects of human perception, substituting low detail models in situations where the difference will not be noticed by the average viewer. Many systems pre-compute a finite number of simplified meshes and swap the appropriate version into the scene where necessary. This is called discrete LOD. Other systems perform mesh simplification dynamically, allowing a mesh to be simplified to any level. This is known as continuous LOD. Figure 2 shows a single model at various levels of detail.

Most LOD research relies on the fact that objects in the distance appear smaller on the screen than those that are close to the virtual camera, and that a person looking at an image of a scene will be less able to discern details on models that take up less screen space. This fact allows the use of different LOD meshes based on an object's distance from the eyepoint. There are other factors, however, that can affect our ability to perceive detail. Lighting, object motion, and surface texture can all help or hinder our ability to distinguish between low and high detail models. Since these factors are visual rather than geometric, an image based simplification scheme is required, such as the one developed by Lindstrom and Turk[7] or the one this project was designed to test.

3 The Masking Effect of Textures

Interactive graphics applications often use surface textures (like wood grain on a table or leopard print on, well, a leopard) on models to increase the realism of a scene. This type of texturing, known as *decal* texture mapping, adds realism to three-dimensional models by pasting a bitmap image onto the surface of the model. The wood grain pattern on the table in Figure 4 is an example of decal texture mapping. Besides making models more visually appealing, texture mapping can have the added bonus of making the faceting of a low detail model less noticeable. This effect is known as *visual masking*. Masking has been studied since the 1950s and applies to other perceptual fields such as auditory processing[14]. Walter et al describe visual masking as "the ability of a (base) visual stimulus to obscure or hide a superimposed (test) stimulus."[12] They further note that this ability depends on a number of factors including the frequencies and contrasts of both the pattern to be hidden, and the applied texture pattern. Figure 3 demonstrates the masking effect of textures of varying contrast, frequency, and orientation on a tessellated cylinder. A full analysis of frequency masking effects would require a thorough discussion of human psychology, physiology, neuroscience, and mathematics. As such it is beyond the scope of this paper, but interested readers should see Ferwerda and Sumanta[14] for a detailed discussion of visual masking in computer graphics.

4 A Simulated Slide Projector

Although the application of textures to a model surface can be a useful way of improving the appearance of low definition models, surface textures are mostly static and are therefore of little use in LODs. It is difficult to think of a real-world situation where an object changes its surface texture dynamically. Our aforementioned leopard, the old adage tells us, is highly unlikely to change his spots. Decal texturing is not the only way to apply texture to a model, however, and it is not the only way to achieve the desired masking effects. Shadows and light projected onto a model can also affect how an image is perceived. An object sitting in the shadow of a tree branch that is swaying in the wind, for example, will be covered in a pattern of light that shifts and changes as the branch moves.

This paper discusses the creation of a model viewer that allows images

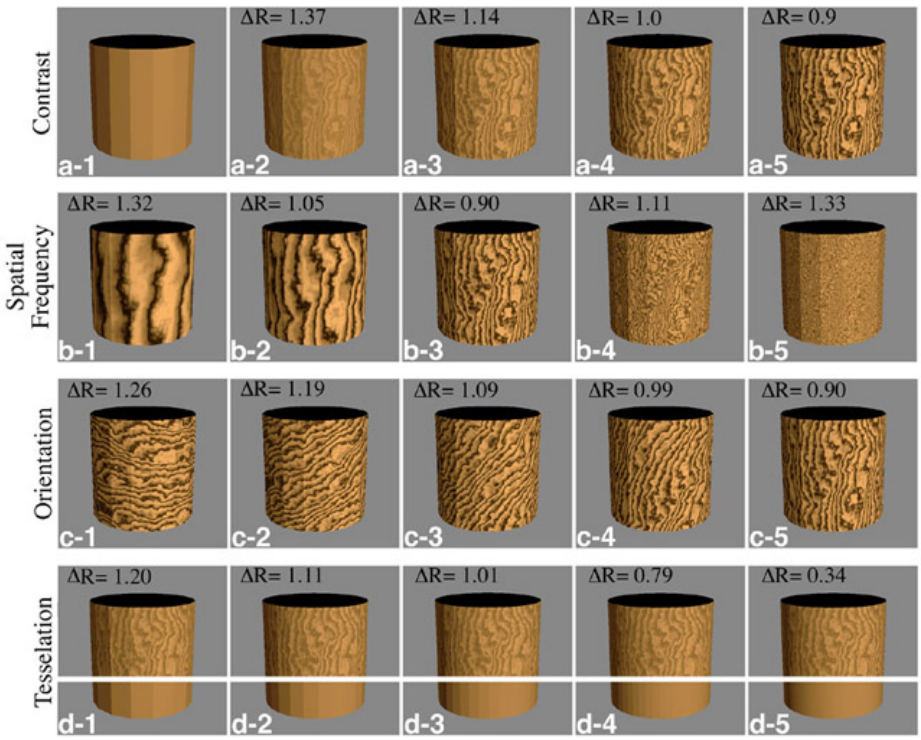

Figure 3: The masking effects of surface textures. From Ferwerda et al[14].

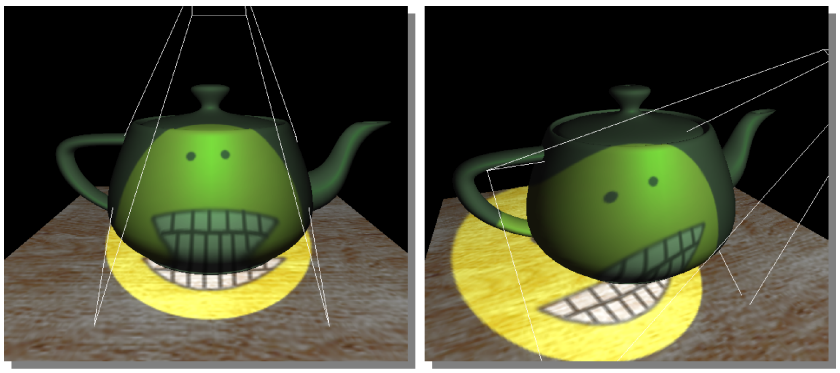

Figure 4: An image showing both decal and projective texturing. From Everitt[5].

to be projected onto an object in order to examine the effect a projected texture has on the perception of that object. This involves creating a virtual slide projector using projective texturing. In order to increase the realism of the images and to make the output more interesting, real time shadow generation is also examined.

The slide projector setup is extremely flexible, allowing any image or animation to be projected onto any object that can be rendered. For example, a spotlight can be easily simulated by projecting an image representing the cross-section of the light's beam[11, 2].

4.1 Texture Projection

Segal et al[11] describe a method called *projective texture mapping* that applies a texture to an arbitrary surface through a perspective-correct projection transform. This method creates the appearance that the texture image has been projected onto the scene from an arbitrary point. The smiley face texture in Figure 4 is applied to the scene using projective texture mapping.

Texture mapping involves taking a point on the surface of an object (in object-space), converting that point into coordinates in the texture image (texture-space), and then drawing the color of our image at that point onto the screen (screen-space). The conversion from object space to texture space is what determines how and where the texture will be mapped to the scene. The conversion process necessary to perform decal texture mapping

is quite straightforward and is built into many graphic environments such as OpenGL.

By using a different method of converting from object space to texture space, we can achieve different texture mapping effects. Imagine that the texture image is sitting somewhere in object space. By projecting the object space points onto that image, we get a conversion that results in the appearance that our texture is being projected onto our object with a slide projector[5]. Since computer graphics already uses projective transforms to convert from object to screen space (we project our scene onto the screen to achieve perspective-correct images) this is generally a very efficient operation. The technique can be used to create fast and convincing lighting effects.

4.2 Projective Textures in OpenGL

OpenGL allows for projective texture mapping with relative ease using the built in texture coordinate generation functions[13, 5, 2]. These facilities take care of the details of converting points from object space to texture space. In OpenGL, there are a number of different methods by which this conversion can take place, allowing for a variety of interesting texture effects including projective texture mapping.

4.3 Shadow Mapping

Shadows help add to the realism of a scene and can help to make the projected texture effect much more pleasing. The lack of shadows in Figure 4 prevents the projected texture from looking like a realistic light projection. A number of real time shadow techniques have been developed including stenciled shadow volumes, planar shadow projection, and shadow mapping. Williams[8] describes a shadow generation scheme that uses the z-buffer to determine the visibility of a polygon from the light's point of view. Because of the use of a two-dimensional image map to calculate shadow placement, this method has become known as *shadow mapping*.

Shadow mapping involves rendering the scene from the light's point-of-view to determine what the light can see so that only those areas are lit. Once this image has been rendered, the z-buffer is saved to memory. This buffer contains the distance from the object rendered to the light for every pixel in the image. The camera is then placed at the eye-point and the scene

is rendered once more. As each pixel in the scene is drawn, the point at that pixel is converted from screen-space to object-space and then to light-space. The x and y components of the converted point are used to look up the depth value in the shadow map, and this shadow map value is compared to the z-value of the light-space point. If they are the same then the light can see the object at that point. If the calculated z-value is greater than the stored value, then the given point is behind some polygon from the light's point-of-view and is therefore in shadow. In Figure 5, s represents the distance from the light to the nearest polygon the light can see, while d represents the distance from the light to the point being rendered. Since the point conversions needed for this process are identical to those needed to project textures, Segal et al[11] suggest using projective texture mapping to do the shadow map comparisons.

To perform the shadow test, the shadow map created in the first step is projected onto the scene from the light's position. If the shadow map value that has been mapped onto a given point is less than the actual distance from the point to the light, then the point is obscured from the light by another polygon and is therefore in shadow. This technique works well in real-time engines, as interactive graphic environments such as OpenGL and Direct3D usually provide depth buffer calculations and projective texturing, and these operations are often hardware accelerated.

5 Shadow Mapping Implementation

Implementing shadow mapping is extremely straightforward using a real-time graphics environment such as OpenGL or Direct3D since many of the required functions are either built-in to the environment, or available through extensions. This section examines the use of shadow maps using OpenGL, but most of the concepts described will work in any graphic environment.

There are 5 steps involved in rendering a scene using a shadow map:

1. creating the shadow map,
2. projecting the shadow map onto the scene,
3. projecting a light-distance texture onto the scene,
4. comparing the shadow map and light distance map, and

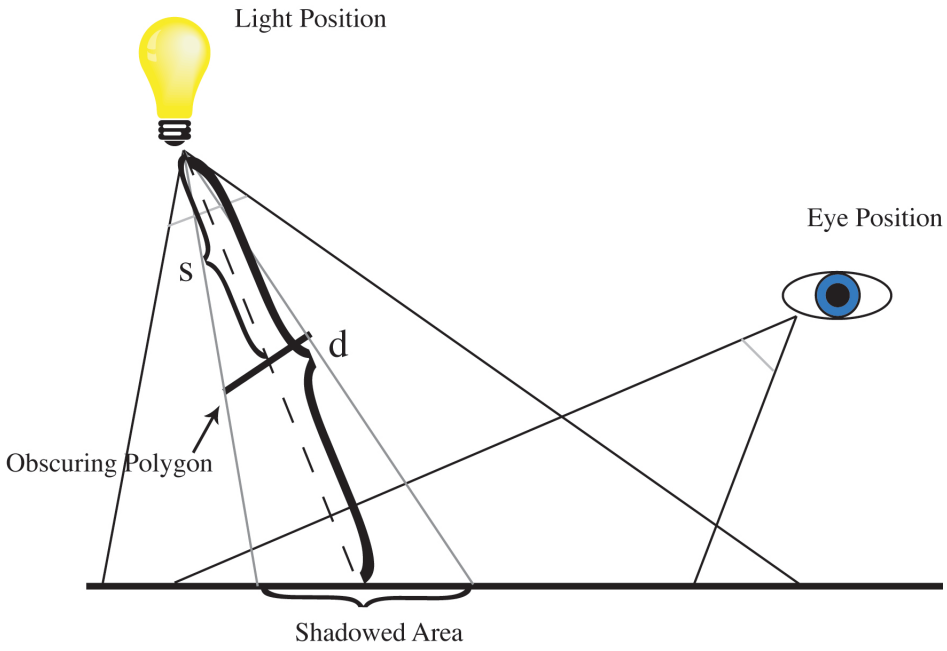

Figure 5: Shadow map comparison. The value s is the distance from the light to the nearest obscuring polygon, and d represents the distance from the light to the shadowed polygon.

5. rendering the scene wherever the two maps are the same.

5.1 Creating the shadow map

Creating the shadow map means rendering the scene from the light's point-of view and extracting the z-buffer from the resulting image[10]. This is entirely straight forward in OpenGL because z-buffers are the default method of hidden surface removal in this environment. The down-side to this approach is the inaccuracy that comes from the limited z-buffer precision on most graphic cards. To overcome this, Cass Everitt[1] suggests projecting the distance values directly onto the scene as alpha values while rendering. This means creating a 1 dimensional texture containing a ramp of alpha values from 0 to 1, and projecting that texture onto the scene such that a point's alpha value represents its distance from the light. Performing this calculation from the light's point of view creates a shadow map in the alpha channel of the frame buffer that contains the distance from the light to every point that is visible from the light's position. In order to maximize the available precision, Everitt also suggests using a cubic alpha ramp for the distance texture rather than a linear one. This means that there are more discrete values for the depth of objects which are closer to the light, improving precision where it is needed most.

Another improvement made by Everitt is to draw the slide image into the unused red, green and blue channels of the shadow map. Then when the final rendering is done, the depth comparisons can be performed using values in the alpha channel while the slide image is projected on the scene in the three color channels. This technique reduces the number of rendering passes needed to produce the final image.

5.2 Rendering the Scene

The last four steps of the algorithm can be done simultaneously using multi-texturing and alpha testing. Alpha testing allows pixels to be drawn or not drawn to the destination buffer based on a function of the values in the alpha channels of the source buffer (in this case the shadow map texture) and/or the destination buffer (in this case the screen). Alpha testing is available in OpenGL and is quite fast.

Multitexturing allows a polygon to be rendered with more than one texture mapped onto it. It is not built into OpenGL, but is available through an

extension. Without multitexturing the scene would have to be rendered once for each texture projected onto the scene. This means one rendering pass to project the shadow map, and one for the light distance map. Multitexturing provides a finite number of texture *units* that can each be configured with a different texture bitmap and mapping scheme[6]. Thanks to the combined slide image/shadow map texture created in the first step, the shadow effect can be accomplished with only two texture units.

The first texture unit contains the same one-dimensional alpha-ramp texture used in creating the shadow map, projected in the same manner. This creates an image map (in the alpha channel of the screen buffer) containing the distance from the light to every point visible from the eyepoint. The second texture unit contains the shadow map, which is also projected onto the scene using a straightforward projective texturing operation. The red, green, blue and alpha values for a given point are a combination of the values calculated by the texture mapping function of each texture unit. The method of combining these values can be adjusted, allowing for various multitexturing effects.

For the slide projector, the color values at each point are simply the product of the values in the color channels in the shadow map (the slide image) and the color of the objects in the scene. This gives us the effect of the slide image being projected onto the objects by simulating the additive nature of light. The alpha values should be the result of a comparison between the value in the shadow map and the value in the distance map at each projected point. This involves performing the *add signed* operator on these two values. If the values in question are a and b, this operation is

$$c = a + (1 - b) - 0.5,$$

which means that c is less than or equal to 0.5 when $a <= b$ and greater than 0.5 when $a > b$.

Rendering a scene with shadows means drawing to the color channels of the screen buffer wherever the alpha value of the shadow map is less than or equal to the alpha value of the distance map. Since the add signed operator ensures that the alpha value of the screen buffer will be less than 0.5 when this is true, drawing shadows is simply a matter of setting up an alpha test to check the whether or not the screen buffer's alpha value is less than or equal to 0.5. The result of rendering the scene with multitexturing and alpha testing enabled is an image with the slide texture projected onto it and realistic shadows.

Simply drawing the scene once in this manner produces a picture in which shadowed pixels are not drawn and are therefore just rendered in the background color. This does not make for a very realistic image as most real-life scenes have some ambient light that illuminates even shadowed areas to some degree. To account for this ambient factor another rendering pass is required. The ambient pass is performed before the slide projector pass, and is done with just ambient lighting enabled. This makes the total number of rendering passes for this effect three: one to create the shadow map, one for the ambient light, and one to create the lighting effect. Two additional passes are also required for each additional spotlight or slide projector that is added to the scene. This makes the technique quite computationally expensive, but it is fast enough to render moderately simple scenes at a reasonable frame-rate using current graphic acceleration hardware.

6 Results

The model viewer developed in this paper was designed as a tool to aid in the demonstration of a mesh decimation technique and, as such, is not suited for an in-depth analysis of visual masking. It does however provide some feel for the factors involved in model error perception.

Figures 6–9 show the output of the viewer with models of varying complexity, under patterns of varying frequency and amplitude. The model in Figures 6 is the base model containing over 69,000 triangles, making it highly detailed, but extremely expensive to render. Figures 7–9 show a number of simplified meshes derived from the original using the QSlim model simplification program[3]. They vary in detail down to the significantly decimated 300 polygon version in Figure 9. The slide pattern projected onto the models in the upper left image of each figure is an image that represents the cross section of a spotlight. This simple pattern does not serve to mask any of the error in the simplified models, and a dramatic difference in quality can be seen between the original mesh and the simpler ones. The slide image in the the other three images in each figure have been modulated using a sin wave of various amplitudes and frequencies in both the x and y directions. This produces the effect of a spotlight shining through some sort of grid or lattice.

The slide used in the lower left of each figure does the best job of hiding the faceting of all of the models. The pattern in this slide has the highest frequency and the greatest amplitude of the three patterned slides. This

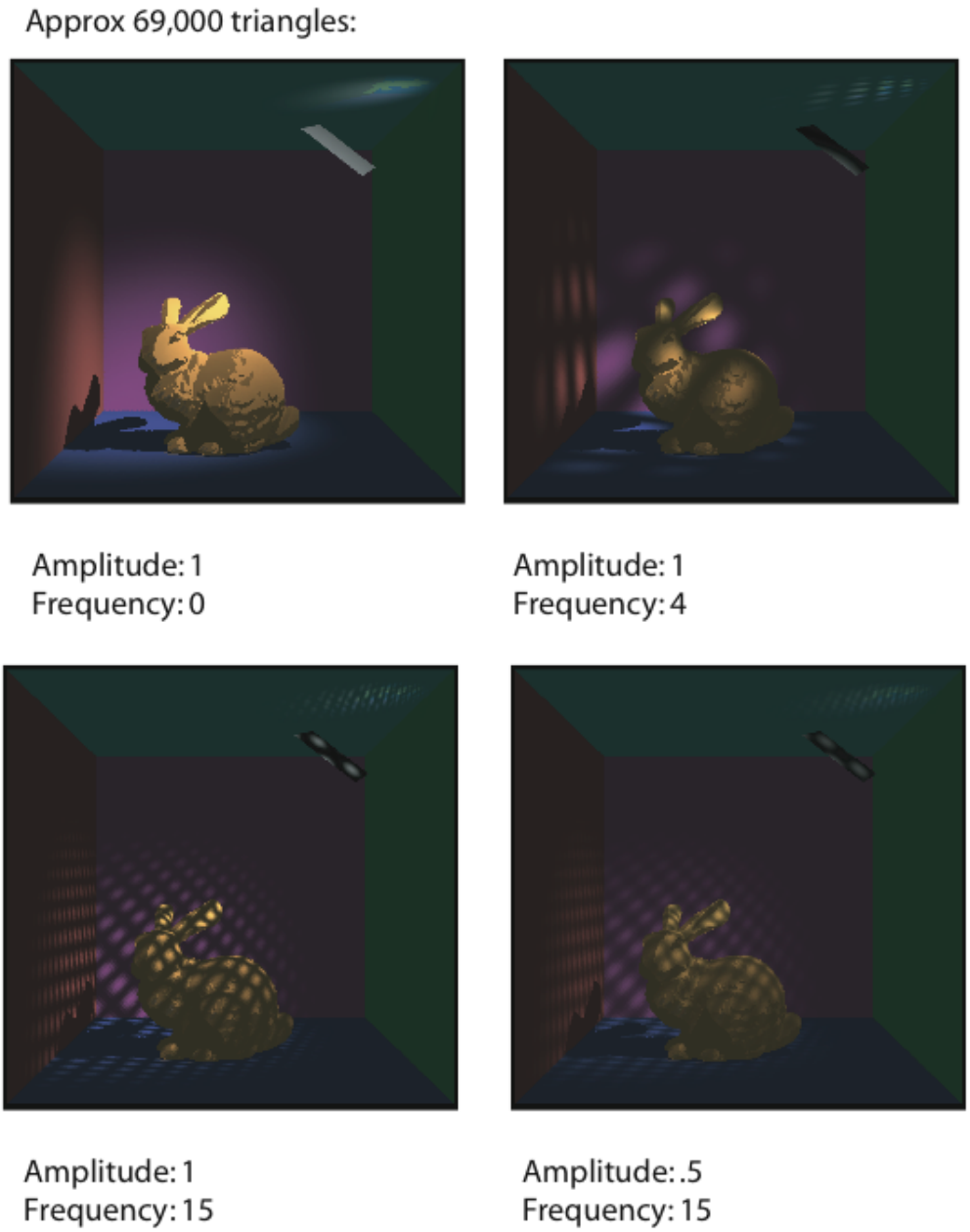

Figure 6: The results of projecting slides of various frequencies and amplitudes onto a mesh containing over 69,000 triangles.

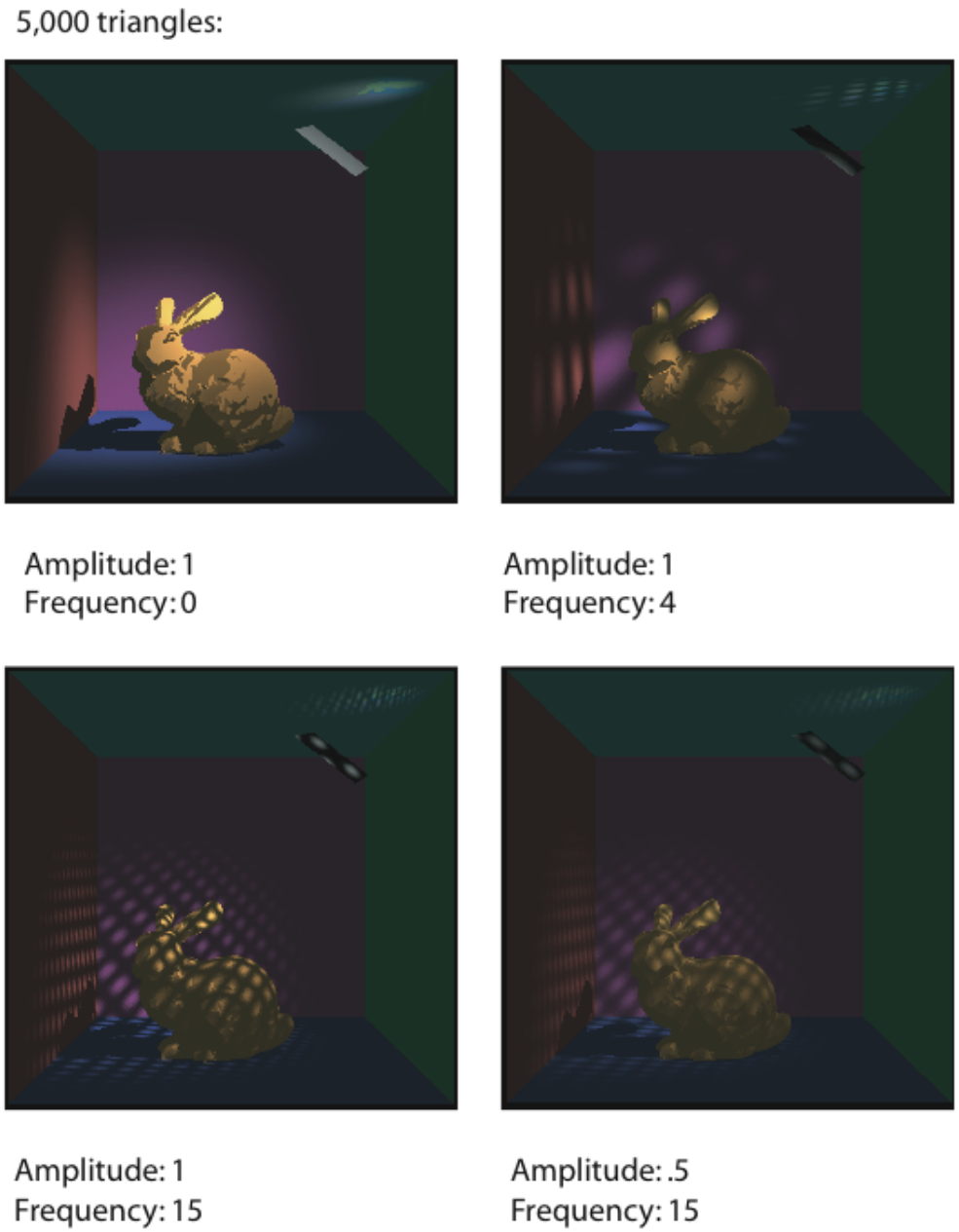

Figure 7: The results of projecting slides of various frequencies and amplitudes onto a mesh containing 5,000 triangles.

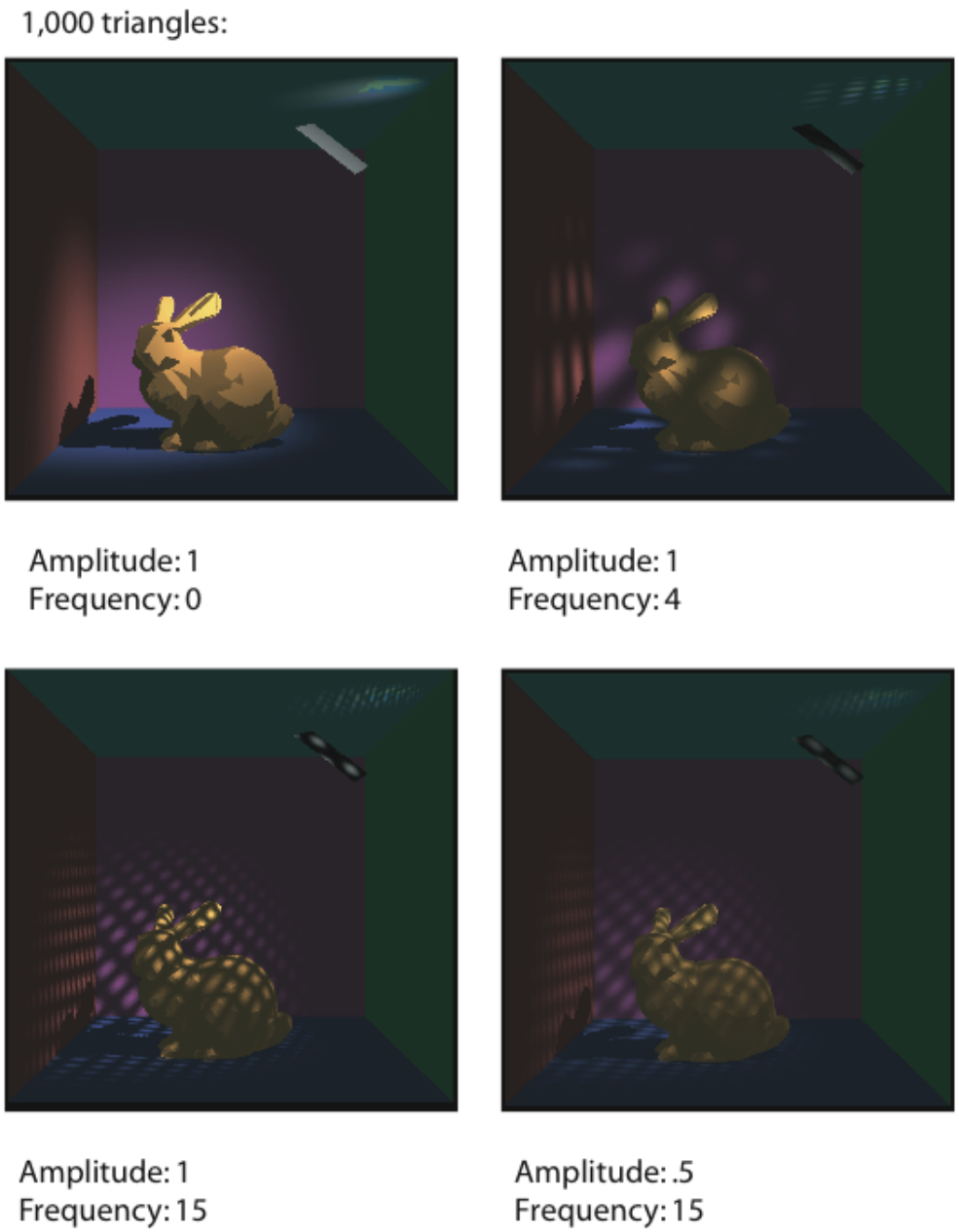

Figure 8: The results of projecting slides of various frequencies and amplitudes onto a mesh containing 1,000 triangles.

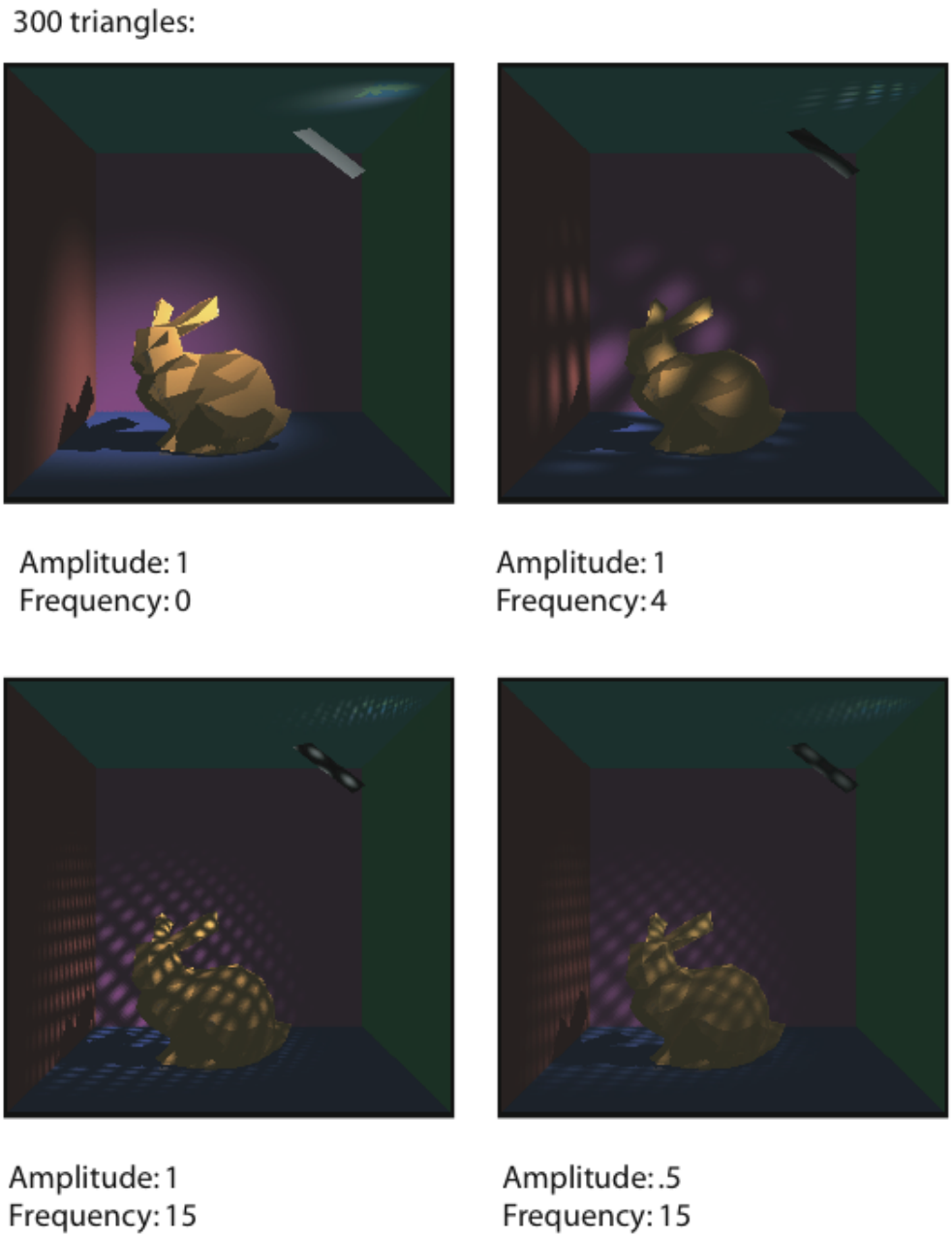

Figure 9: The results of projecting slides of various frequencies and amplitudes onto a mesh containing 300 triangles.

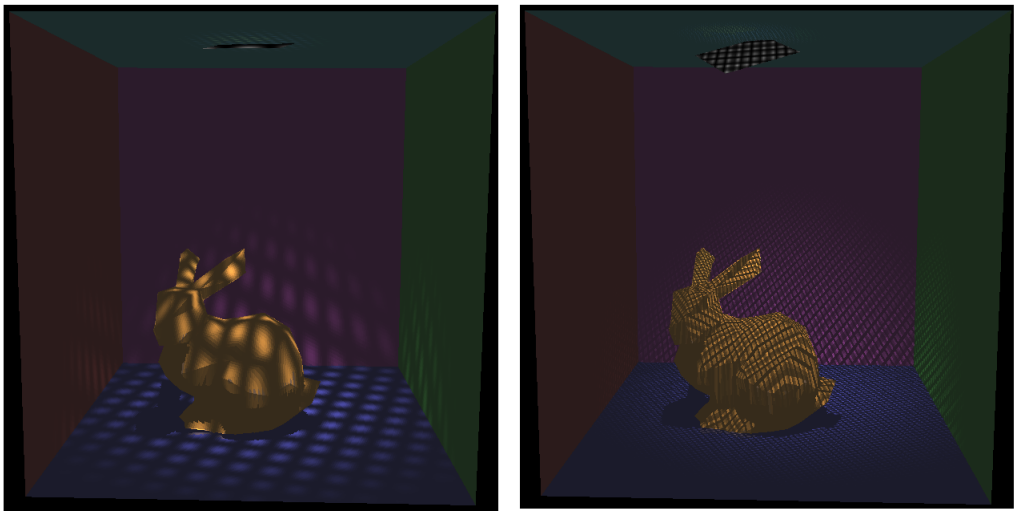

Figure 10: Two different textures applied to a mesh of 300 polygons. The texture on the left does a better job of masking error.

suggest that patterns with higher frequencies and amplitudes are better able to mask unwanted artifacts than patterns with lower frequencies and amplitudes. This is not always the case, and Ferwerda et al note that visual masking is a function of the amplitude, frequency and orientation of both the pattern being masked and the masking pattern[14]. Figure 10 demonstrates this fact using the 300 triangle mesh. The low frequency pattern projected onto it in the left-hand image does a much better job hiding the faceting of this mesh than the high frequency pattern in the right-hand image. Figure 11 shows that the opposite is true for the 5000 triangle mesh.

These results show that an LOD system could adjust the polygon count of a model based on the pattern of light and shadows that are projected onto it. Such a system could improve performance by using simple models when the scene allows, while maintaining visual fidelity when it doesn't. These images also indicate that deciding how much to decimate a model given a projected pattern is not entirely straightforward. In order to build an LOD systems such as this, a comprehensive model of visual masking like the one described in [14] would be necessary, and this model would need to be fast enough to run in real-time.

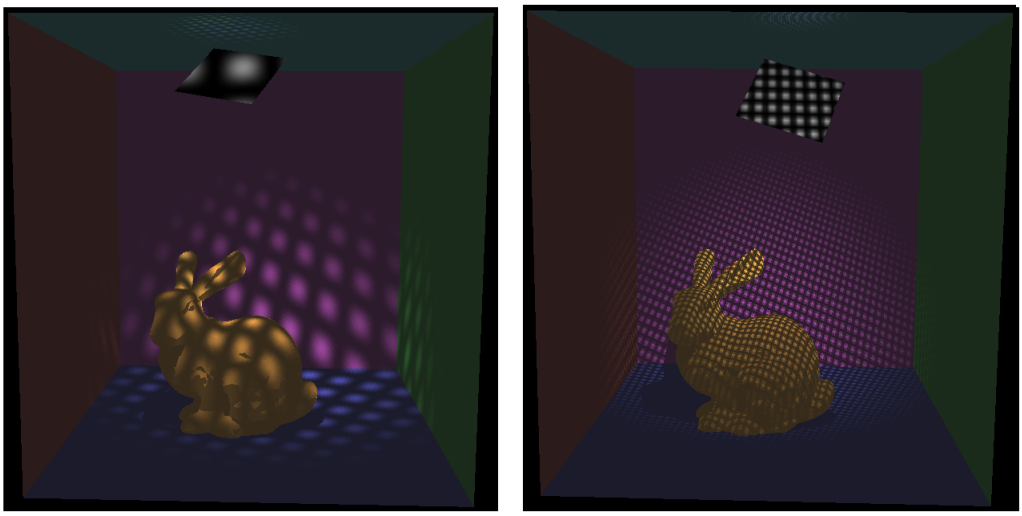

Figure 11: Two different textures applied to a mesh of 5000 polygons. The texture on the right does a better job of masking error.

7 Conclusion

Shadows are extremely important to human perception. Though we rarely notice them when they are present, their absence in an image can be extremely conspicuous. Photographers and film makers have long known the power of shadows and often use them to impart meaning and to play with their audiences emotions. Soft defining shadows can help bring out the shapes of a still life, while the stark, eerie shadows of a moonlit night can help set the mood for a horror film. The addition of shadows can have an important affect on our response to a computer generated image too.

With the increasing popularity of video games, and the growing use of three-dimensional interactive graphics in everything from scientific analysis to engineering, real-time graphics have become an extremely important part of computer science research. Because of the need for high frame rates in interactive graphics, realism must be balanced with efficiency, and squeezing every ounce of performance out of a both software and hardware is a must for cutting edge graphics. Though reasonably fast, shadow generation techniques are often too expensive to be used extensively in games and simulations.

LODs are a great way of increasing performance without significantly

compromising realism. This performance boost can allow the addition of special effects such as real-time shadows to interactive graphics. As this paper shows, the addition of shadows to a scene could actually help improve the performance of LOD systems by masking the error they produce. Shadow mapping requires multiple rendering passes over the geometry of our scene, but if it also allows us to reduce the complexity of that geometry, the performance degradation it causes may be somewhat mitigated. As graphics hardware improves and software techniques are refined, real-time shadows will surely become an integral part of immersive games and simulations. Image-based level of detail reduction is also likely to grow in importance. Combined, these concepts have the potential to vastly improve the way we render scenes, and change the look of real-time graphics for the better.

8 Source-Code and Demo

The source-code for the model viewer described in this paper, as well as a Windows binary are available online at:

http://www.ronandowling.com/shadow/

References

[1] Cass Everitt. Neato. From the Nvidia SDK.

[2] David G Yu. projtex.c. From SIGGRAPH '97 Advanced OpenGL Programs.

[3] Michael Garland. Qslim 2.0. Software package, March 1999.

[4] Hugues Hoppe. Progressive meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[5] Cass Everitt. Projective texture mapping. Whitepaper from the Nvidia SDK.

[6] Michael Gold. The arb_multitexture extension. Technical Pesentation at GDC '99, 1999.

[7] Peter Lindstrom and Greg Turk. Image-driven simplification. *ACM Transactions on Graphics*, 19(3):204–241, 2000.

[8] Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics*, 12(3):270–274, August 1978.

[9] Mark J. Kilgard. Shadow mapping with todays opengl hardware. *CDC 2001*, 2001.

[10] Cass Everitt, Ashu Rage, and Cem Cebenoyan. Hardware shadow mapping. Whitepaper.

[11] Mark Segal et al. Fast shadows and lighting effects using texture mapping. *Proceedings of SIGGRAPH '92*, pages 249–252, 1992.

[12] Bruce Walter. Using perceptual texture masking for efficient image synthesis. *EUROGRAPHICS 2002*, 21(3), 2002.

[13] J. Neider et al. *OpenGL Programming Guide: The Official Guide to Learning OpenGL.* AddisonWesley, Reading Mass., 1st edition, 1993.

[14] James A. Ferwerda, Sumanta N. Pattanaik, Peter Shirley, and Donald P. Greenberg. A model of visual masking for computer graphics. *Computer Graphics*, 31(Annual Conference Series):143–152, 1997.