EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (2002), Number 3

Using Perceptual Texture Masking for Efficient Image Synthesis

Bruce Walter[†] Sumanta N. Pattanaik[‡] Donald P. Greenberg[†]

[†]Cornell University [‡]University of Central Florida

Abstract

Texture mapping has become indispensable in image synthesis as an inexpensive source of rich visual detail. Less obvious, but just as useful, is its ability to mask image errors due to inaccuracies in geometry or lighting. This ability can be used to substantially accelerate rendering by eliminating computations when the resulting errors will be perceptually insignificant.

Our new method precomputes the masking ability of textures using aspects of the JPEG image compression standard. This extra information is stored as threshold elevation factors in the texture's mip-map and interpolated at image generation time as part of the normal texture lookup process. Any algorithm which uses error tolerances or visibility thresholds can then take advantage of texture masking. Applications to adaptive shadow testing, irradiance caching, and path tracing are demonstrated.

Unlike prior methods, our approach does not require that initial images be computed before masking can be exploited and incurs only negligible runtime computational overhead. Thus, it is much easier to integrate with existing rendering systems for both static and dynamic scenes and yields computational savings even when only small amounts of texture masking are present.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

Keywords: *Visual Perception, Perceptual Rendering, Texture Mapping, JPEG*

1. INTRODUCTION

In the 25 years since their introduction, texture maps [2] have become an indispensable part of image synthesis and a required feature in modern rendering systems. They are widely used to represent fine visual detail and produce more realistic looking images. Just as valuable is their ability to hide, or mask, image artifacts (e.g., due to approximations in the geometry or illumination) that would otherwise be visible. Modelers have long used this effect in an ad hoc manner by adding or modifying textures to hide rendering artifacts.

If it can be exploited systematically and efficiently, this same ability can be used to accelerate rendering algorithms. Rendering algorithms often waste effort on illumination components whose effect is too small for human viewers to perceive. This is especially true for textured regions where the threshold of visibility is generally higher. This represents both a challenge and an opportunity to efficiently identify and eliminate such unnecessary computation.

In this paper we present a new method to inexpensively find visual error tolerances that include masking due to texture. Unlike prior methods, our threshold computations incur only negligible runtime overhead and do not require the computation of initial approximate images. Only a simple preprocess of each texture map is required, which can be done offline and is based on elements of the well-known, widely-used JPEG image compression standard [20]. Threshold elevation factors are stored in a standard mip-map struc-

[†] Program of Computer Graphics,
580 Rhodes Hall, Cornell University, Ithaca, NY, 14853, USA
[‡] University of Central Florida, SEECS,
Department of Computer Science, Room 250,
Orlando, FL, 32816-2362, USA

ture along with the texture and can be rapidly accessed as part of the normal texture lookup process. Thus our method can easily be added to existing rendering systems.

To show its generality, we have applied our method to accelerate three different rendering algorithms: adaptive shadow testing, irradiance caching, and path tracing.

2. PREVIOUS WORK

Many researchers have used visibility thresholds as the termination criteria for rendering computations. The simplest and most common approach is to use Weber's law, which states that the visibility threshold is a fixed fraction of the base luminance. However, while humans can sometimes perceive luminance changes of less than 1%, visibility thresholds are typically much larger in textured regions.

Drawing on results from the perception literature, Ferwerda et al. [6] presented the most comprehensive model of masking in computer graphics. However, their method requires an exact reference image as input, making it impractical for use in rendering acceleration. Ramasubramanian et al. [18] modified their technique to use an approximate image instead, but their approach has not been widely adopted due to the high computational overhead and lack of solid perceptual validation.

Bolin and Meyer [3, 4] use perceptual models, including masking, to control adaptive sampling [17] in the image plane. They initially used some JPEG-like operations [3], but later switched to a more sophisticated visual model [4]. Since their visual model is updated after each sample is taken, limiting computational overhead was a major concern in its design. Volevich et al. [19] also use a detailed perceptual model [15], but use it only for algorithm selection due to its high cost, and thus do not exploit masking effects. Myszkowski[16] applied an enhanced model to the generation of animation sequences, but only uses the perceptual model on a small subset of the frames to reduce its overhead.

Unlike these methods, we have not tried to include the evaluation of a full perceptual model during image generation. Instead we concentrated exclusively on texture masking and providing a way for existing algorithms to take advantage of the presence of textures with minimal modifications and negligible overhead.

Perception models that do not include texture masking are generally inexpensive and have been used to accelerate various algorithms (e.g.,, radiosity[7, 8]). In this work we assume such a perception model is already in use, and we concentrate solely on computing the additional elevation in visibility thresholds caused to the presence of texture maps.

Dumont et al.[5] considered texture masking in their cost-benefit model for managing the texture cache associated typical graphics hardware. They provided techniques for estimating the perceptual cost of using lower resolution versions of a texture when computing a image. Yeeet al. [25] extended a perception model to include time-dependent effects such as attention and tracking, which are not included in this paper.

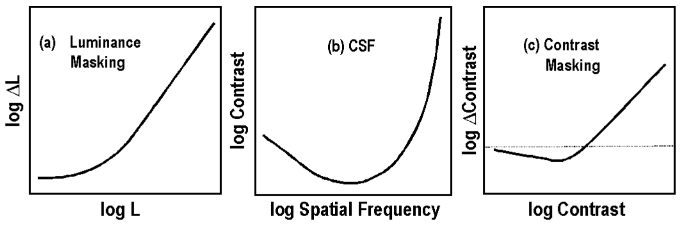

Figure 1: *(a) Visibility threshold (ΔL) vs. base luminance L for uniform stimuli. The roughly linear pattern is known as Weber's law [9]. (b) Visibility thresholds ($\Delta L/L$) for test stimuli of varying frequency against a uniform base stimuli. Notice there is a large increase in visibility thresholds at high frequencies. This plot is the inverse of the well known contrast sensitivity function (CSF) (c) Visibility threshold vs. base contrast for stimuli of a similar frequency. The relation between threshold for target and base pattern contrast is similar to the Weber's Law relation shown in (a). See [18] and references for more detail.*

3. VISUAL MASKING

The ability of a (base) visual stimulus to obscure or hide a superimposed (test) stimulus is called *visual masking* [6]. The degree of masking depends on many factors including the intensities and spatial frequencies of the stimuli. For a fixed base stimulus, the intensity at which the test stimulus becomes noticeable is called the *visibility threshold*. Sample plots of how the visibility threshold varies with intensity and frequency are shown in Figure 1. The observation that visibility thresholds vary roughly linearly with the intensity of the base stimulus is known as Weber's law [9]. Thresholds increase significantly for higher spatial frequencies and stronger base contrasts.

3.1. Exploiting Masking by Textures

Masking can be used to hide approximation errors and accelerate rendering algorithms. The visibility threshold can be used as a quantitative measure of the amount of error than can be visually tolerated in any rendered image. Textures are important ingredients in many synthetic scenes and often the most important sources of high frequency and high contrast patterns and hence exhibit a high degree of visual masking. By precomputing and storing visibility threshold factors with each texture, we can take advantage of texture masking with extremely low overhead at image generation time. Moreover, each texture need only be preprocessed once, regardless of how many times it is used.

We cannot simply store visibility thresholds or error tolerances as these depend on the base intensity, which is not known until the illumination is computed. Instead we follow Ramasubramanian [18] and store threshold elevation factors, or the multiplier by which the threshold is raised by the presence of the texture.

Another difficulty is that the visual frequency content of a texture map changes as a function of viewing distance and angle. It would be impractical to store values for every possible viewing configuration. Instead we precompute elevation factors for the texture at a fixed set of resolutions and interpolate at image generation time from the resolutions that most closely match the texture's apparent resolution in the image plane. This can be done quickly and easily using the mip-map[24] pyramidal scheme. For systems that already support mip-mapping for texture anti-aliasing, the elevation factor lookup can be done as part of the normal texture lookup procedure.

Compared to previous visual masking computation methods, this approach is orders of magnitude less expensive because it does not require the computation of initial images or evaluation of visual models during image generation. While we do not include masking due to occlusion (e.g., silhouette edges) or illumination (e.g., shadow boundaries) changes, these are frequently less important sources of masking and much more expensive to compute.

4. TEXTURE PREPROCESSING

For each texture and each resolution in its mip-map representation, we need to compute the corresponding threshold elevation map. This computation requires a method to find the various frequency components in the texture and their contrast levels, and then a mathematical model to convert frequency and contrast information to visibility thresholds.

While we initially experimented with using a Laplacian pyramid approach similar to [18], we have instead used an approach based on elements of the JPEG compression standard [20, 10] with some extensions. This allows us to draw on the wealth of experience, trust, and optimized code associated the JPEG standard, or more specifically, the DCT-based lossy image compression which is the most widely used part of the JPEG standard.

We should note that there is also a more recent, but not yet widely used standard called JPEG 2000 which is based on wavelet transforms instead of the DCT. The new standard's main advantage is that it produces less objectionable artifacts at higher compression ratios where lossy compression causes image changes significantly above visibility thresholds. However, our interest is in estimating errors thresholds at or below visibility.

4.1. Some JPEG Basics

The first step in JPEG encoding is the transformation of the color values from RGB to a luminance/chrominance space. Because the human visual system is more sensitive to changes in luminance than in chrominance, we will only consider the luminance channel processing in computing our error thresholds. Next, the image is divided into 8x8 blocks and each block is transformed using a discrete cosine transform (DCT) which converts it from the spatial to the frequency domain. The first coefficient of the transformed block is proportional to the average luminance within the block and the other coefficients reflect the content of successively higher frequency components. Typically many of the coefficients will be near zero.

Next a quantization matrix is applied to reduce the precision (and hence bits) with which these coefficients are stored. Quantization is the only lossy step in JPEG compression (where error is introduced); the rest of the encoding is lossless and not relevant for us. During decoding, this process is reversed, producing frequency coefficients which are, in general, different from the original ones. After applying the inverse DCT, we can get a luminance block which is perceptually very close to the original if the quantization matrix was well chosen.

We can express this using matrix algebra as follows:

$$F = TYT^T \tag{1}$$

$$|F^{\dagger}_{i,j} - F_{i,j}| \leq \frac{1}{2}Q_{i,j} \tag{2}$$

$$Y^{\dagger} = T^T F^{\dagger} T \tag{3}$$

where Y is the 8x8 matrix of original luminance values, T is the matrix form of the 8x8 DCT shown below, and F is the matrix of frequency coefficients. $F^{\dagger}$ is the perturbed matrix of coefficients obtained after decompression due to the quantization process. Its maximum deviation from the original F is controlled by the quantization matrix Q as shown in Equation 2. Applying the inverse DCT transform to $F^{\dagger}$ yields the decompressed luminance values $Y^{\dagger}$.

$$T_{i,j} = \begin{cases} \frac{1}{2\sqrt{2}} & \text{if } i = 0 \\ \frac{1}{2}\cos\left(\frac{(2j+1)i\pi}{16}\right) & \text{otherwise} \end{cases} \tag{4}$$

The quantization values should be below the masking threshold for the appropriate frequency and luminance content of the image to preserve image quality. Masking thresholds are dependent on the image content. Luminance-dependent masking depends on the local intensity and the frequency-dependent masking depends on the contrast in the frequency component of the image block. Empirical relationships between the image content and the threshold have been established from subjective experiments [14, 1]. Strictly following this relationship would result in a different quantization matrix for each image block. For simplicity, a single image independent quantization matrix is usually used in standard JPEG compression.

Although the JPEG standard does not mandate the use of

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 1: *The luminance quantization matrix from Annex K of the JPEG standard* [10]*. The standard allows any quantization matrix to be used, but most encoders use this matrix or scaled versions of it. Each row represents coefficients for increasing horizontal frequencies and each column for increasing vertical frequencies. These coefficients reflect the threshold vs. frequency relation shown in Figure 1. Thus this quantization matrix accounts for threshold elevation due to frequency.*

a particular quantization matrix, most algorithms use the luminance quantization table shown in Table 1 from Annex K of the standard [10]. The values are derived from the assumption that the average image luminance is that of a mid-gray level. For an 8-bit image with a mid-gray value of 128, the 8x8 DCT transform produces a DC component of 1024. The DC component of the quantization matrix, $Q_{0,0}$, contains the value 16, which is roughly 1.5% of the mid-gray DC level as we would expect from Weber's Law without frequency components (see Figure 1a). The other elements of the quantization matrix vary with the visibility thresholds for successively higher frequency components against a mid-gray background signal in the way we would expect from Figure 1b.

4.2. Basic Algorithm

Although JPEG is lossy and introduces error into images, it is widely accepted because at moderate compression rates those errors are largely imperceptible. Intuitively, it should be equally acceptable to allow errors of a similar magnitude during rendering, especially if this can result in significant computational savings.

To compute the visibility threshold for a texture at a particular resolution, we convert the RGB texel values to luminances and apply a DCT transform just as in Equation 1. Next we create a matrix of perturbed frequency components as follows:

$$F^{\dagger}_{i,j} = \begin{cases} F_{i,j} & \text{if } F_{i,j} < Q_{i,j} \\ F_{i,j} + \frac{1}{2}\text{sign}(F_{i,j})Q_{i,j} & \text{otherwise} \end{cases} \tag{5}$$

Note that these values are within the maximum deviation allowed in the JPEG encoding/decoding process (See Equation 2).

We do not perturb frequency components which are below the quantization threshold, because such components are assumed not visible and hence do not contribute to masking. Moreover, the presence of subthreshold patterns can actually increase the eye's sensitivity to similar patterns (Figure 1c).

After performing the inverse DCT on $F^{\dagger}$ to get $Y^{\dagger}$, we could get the visibility threshold at every pixel from $|Y^{\dagger} - Y|$. However, the magnitude of the error threshold depends on the illumination level which will not be known until image generation time. Instead we compute the ratio of this error threshold to the error threshold without including frequency masking (i.e. excluding texture effects). According to Weber's law, both thresholds should scale roughly linearly with illumination level, so their ratio is roughly independent of it. This second threshold can be computed by perturbing the DC component only (i.e. $F_{0,0}$) or equivalently by perturbing the pixel values $Y_{i,j}$ directly. The visibility threshold elevation factor c is given by:

$$c_{i,j} = \max\left[1,\ \frac{|Y^{\dagger}_{i,j} - Y_{i,j}|}{\frac{1}{2}Y_{i,j}\frac{Q_{0,0}}{1024}}\right] \tag{6}$$

where the denominator is derived in the same way as the numerator except that we perturb only the DC component instead of all its components as in Equation 2. Equivalently we can notice that $Q_{0,0}/1024$ is relative visibility threshold based solely on Weber's Law without frequency considerations. (As mentioned before 1024 is the DC component of the mid-gray level 8-bit image block which was assumed as the average image luminance in the design of the JPEG standard).

We now have a complete algorithm for computing elevation factors. It is relatively inexpensive, roughly equivalent to one JPEG compresssion/decompression cycle, and can use the optimized DCT routines available on many platforms. For non-transparent textures, the elevation factors (typical range 1-16) can be quantized and stored in the alpha channel, and otherwise we just add one additional channel to each texture and mip-map.

4.3. Adaptive Quantization Matrices

Standard JPEG uses a single quantization matrix over an entire image. While adaptive quantization matrices have been proposed as an extension to the standard, they are not generally used because of the potential storage overhead. This is not a problem for us, and using adaptive matrices allows us to better model masking.

We follow an approach similar to the one suggested by Watson [23] and model local adaptation by modifying Q according to the local average intensity over the block to get a new quantization matrix,Q^a, which more accurately reflects Weber's law.:

$$Q^a_{i,j} = \frac{F_{0,0}}{1024}Q_{i,j} \tag{7}$$

The matrix still does not take proper advantage of the fact that stronger frequency signals can mask larger errors as shown in Figure 1c. We create a new quantization matrix Q^m that takes into account the actual frequency content in each block as:

$$Q^m_{i,j} = \begin{cases} Q^a_{0,0} & \text{if } i = j = 0 \\ Q^a_{i,j} \max\left[1, \left|F_{i,j}/Q^a_{i,j}\right|^{0.7}\right] & \text{otherwise} \end{cases} \quad (8)$$

The exponent value of 0.7 agrees with the contrast masking measurements made by Legge [13]. Substituting Q^m for Q in the basic algorithm gives us better threshold elevation factors at minimal cost. A MATLAB implementation of the algorithm has been given in the Appendix.

5. RESULTS

The threshold elevation factors for several sample textures are shown in Figure 2 along with the average elevation factor for each. As expected, the largest elevation factors correspond with the regions of high frequency and contrast. However, all these textures contain enough masking to produce significant computational savings.

As an example of these savings, we have adapted three different rendering algorithms to use our elevation maps and measured the performance gains when rendering the scene from the viewpoint shown in Figure 3. This is an environment with 65636 polygons and 44 light sources. Remember that the savings are strongly dependent on the scene, viewpoint, and rendering algorithm, so our results should be taken as a rough indication only. Since the example scene contains several image regions with little or no texture masking such as the untextured couch and the ceiling with its low contrast and frequency texture, we should not expect overly dramatic speedups.

We first computed a direct-plus-ambient solution using a simplified version of Ward's adaptive shadow testing for ray tracing [21]. At each pixel, the light sources are sorted in descending order of potential contribution, and successively evaluated using shadow rays to check visibility. The computation is terminated if the remaining unevaluated potential contribution falls below an error threshold. For scenes with many lights, this results in fewer shadow rays and large savings. Using adaptive shadow testing and an error threshold of 1.5%, which is roughly the visibility threshold for untextured surfaces, we reduced the computation time from 50 to 13.5 seconds. Modulating the threshold by our elevation maps, however, further reduced the computation time from 13.5 to 8 seconds for a speedup of 1.7 times with no loss of image quality.

Irradiance caching [22] accelerates indirect illumination computations by caching and reusing expensive irradiance samples. New samples are only computed when no previous sample is sufficiently close, and otherwise inexpensive interpolation is used. The density of these irradiance samples is controlled by the spacing parameter a. Because Ward showed that there is a roughly linear relationship between interpolation error and his spacing parameter a, we can use our error elevation factors to modulate the a parameter at each pixel. The result is that fewer irradiance samples are needed in textured regions. We used this to accelerate the gathering of diffuse indirect illumination from a photon map [11]. Using our elevation factors reduced the total number of irradiance samples needed from 20200 to 11200, with a corresponding speedup of 1.8.

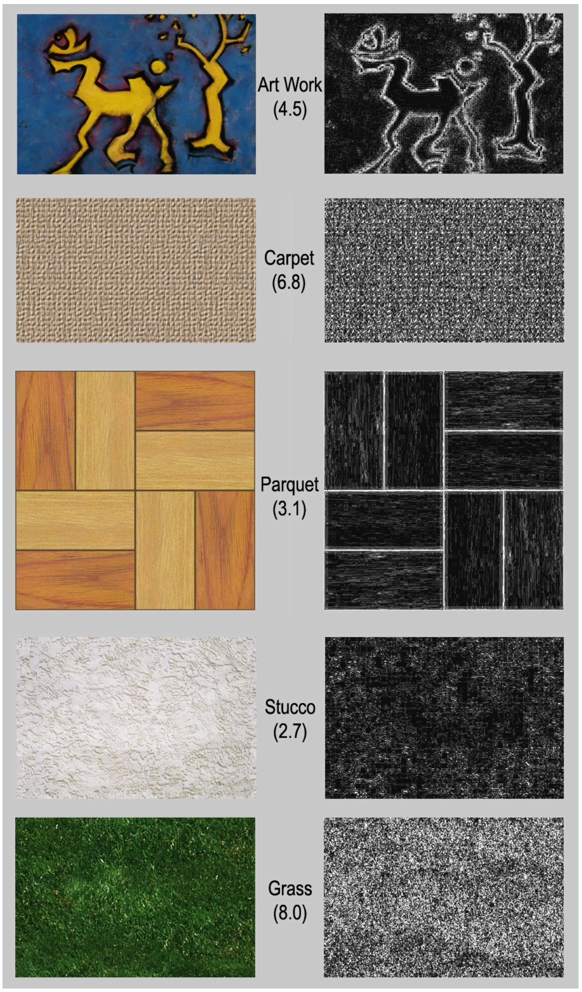

Figure 2: *Several sample textures and their corresponding threshold elevation factor maps. The average elevation for each texture is shown in parentheses. Note that a separate elevation map is computed for each level in a texture's mip-map, although we have shown only the highest resolution level map here.*

While path tracing[12] is not the most efficient way to compute global illumination, it was used as an example application in previous approaches [18, 4]. We implemented a path

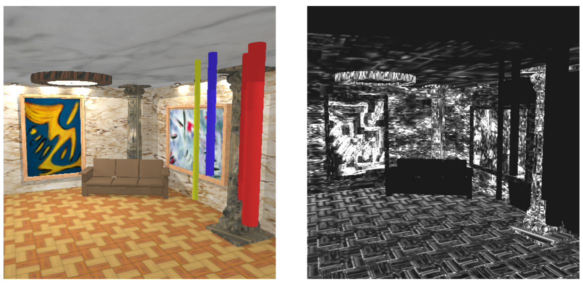

Figure 3: *A direct-plus-ambient rendering of the scene and viewpoint used in our examples. The threshold elevation factors for this view are shown on the right. The average threshold elevation factor is 2.7.*

tracer which, at each pixel, traces paths until the estimated standard deviation falls below an error threshold. Starting with a base threshold of 1.5%, we computed images of our scene both with and without using threshold elevation factors. The resulting speedup was 2 times, which although respectable, is much less than that reported by [18]. However, they compared against a non-adaptive path tracer, while our base case already contains adaptive sampling due to variance estimation and Weber's law. If we similarly compare against a non-adaptive path tracer, our speedup is 15 times.

6. CONCLUSIONS

We have presented a new technique for exploiting the additional rendering error that can be tolerated in texture-mapped regions. We do not claim that this method is more complete or more accurate than previous methods, but, it is less expensive, simpler, and easier to integrate into existing systems. Thus it can be used to accelerate time-critical computations like our direct illumination example, where the overhead of previous methods would be prohibitive.

For systems which already support texture mapping and mip-map texture anti-aliasing, the additional implementation overhead is minimal: the storage of one additional channel per texture, a simple preprocess per texture using JPEG-like operations, and the trilinear interpolation of the error threshold elevation factors along with the normal texture channels during texture lookup. With these components, virtually any algorithm that uses per-pixel error thresholds can be accelerated.

Because our error threshold elevation factors are based on the widely used JPEG compression standard, we gain the benefit of the collective knowledge, experience, and optimized code associated with it. We can also potentially leverage future research and advances in perceptually-based image compression.

Acknowledgments

Thanks to Hector Yee for providing the room model to our anonymous reviewers for their helpful comments. This work was supported by Intel Corporation and the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219).

References

1. A. J. Ahumada, Jr. and H. A. Peterson. Luminance model based dct quantization for color image compression. In B. E. Rogowitz, editor, *Visual Processing and Digital Display III*. 1992.
2. J. Blinn and M. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, October 1976.
3. M. R. Bolin and G. W. Meyer. A frequency based ray tracer. *Computer Graphics (ACM Siggraph '95 Conference Proceedings)*, pages 409–418, August 1995.
4. M. R. Bolin and G. W. Meyer. A perceptually based adaptive sampling algorithm. *Computer Graphics (ACM Siggraph '98 Conference Proceedings)*, pages 299–310, July 1998.
5. R. Dumont, F. Pellacini, and J. A. Ferwerda. A perceptually-based texture caching algorithm for hardware-based rendering. In *Rendering Techniques '01*, pages 249–256. Springer-Verlag/Wien, 2001.
6. J. A. Ferwerda, S. N. Pattanaik, P. Shirley, and D. P. Greenberg. A model of visual masking for computer graphics. *Computer Graphics (ACM Siggraph '97 Conference Proceedings)*, pages 143–152, August 1997.
7. S. Gibson and R. Hubbold. Perceptually-driven radiosity. *Computer Graphics Forum*, 16(2):129–140, June 1997.
8. D. Hedley, A. Worrall, and D. Paddon. Selective culling of discontinuity lines. In *Rendering Techniques '97*, pages 69–80. Springer-Verlag/Wien, 1997.
9. D. C. Hood and M. A. Finkelstein. Sensitivity to light. In Boff, Kauffman, and Thomas, editors, *Handbook of Perception & Human Performance*, chapter 5. 1986.
10. Digital compression and coding of continuous-tone still images. ISO/IEC 10918, Feb 1994. JPEG Standard.
11. H. W. Jensen. Global illumination using photon maps. In *Rendering Techniques '96*, pages 21–30. Springer-Verlag/Wien, 1996.
12. J. T. Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, August 1986. ACM Siggraph '86 Conference Proceedings.
13. G. E. Legge. A power law for contrast discrimination. *Vision Research*, 21:457–467, 1981.
14. H. Lohscheller. A subjectively adapted image communication system. *IEEE Trans. Communications*, 32:1316–1322, 1984.
15. K. Myszkowski. The visible differences predictor: Applications to global illumination problems. In *Rendering Techniques '98*, pages 223–236. Springer-Verlag/Wien, 1998.

16. K. Myszkowski, T. Tawara, H. Akamine, and H.-P. Seidel. Perception-guided global illumination solution for animation rendering. *Computer Graphics (ACM Siggraph '01 Conference Proceedings)*, pages 221–230, August 2001.

17. J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. *Computer Graphics*, 23(3):281–288, July 1989. ACM Siggraph '89 Conference Proceedings.

18. M. Ramasubramanian, S. N. Pattanaik, and D. P. Greenberg. A perceptually based physical error metric for realistic image synthesis. *Computer Graphics (ACM Siggraph '99 Conference Proceedings)*, pages 73–82, August 1999.

19. V. Volevich, K. Myszkowski, A. Khodulev, and E. A. Kopylov. Using the visual differences predictor to improve performance of progressive global illumination computation. *ACM Transactions on Graphics*, pages 122–161, April 2000.

20. G. K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, pages 30–44, April 1991.

21. G. Ward. Adaptive shadow testing for ray tracing. In *Proceedings of the Second Eurographics Workshop on Rendering (Barcelona, May 1991)*, 1991.

22. G. J. Ward, F. M. Rubinstein, and R. D. Clear. A ray tracing solution for diffuse interreflection. In J. Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 85–92, Aug. 1988.

23. A. B. Watson. Perceptual optimization of dct color quantization matrices. *Proc. of International Conf. on Image Processing*, pages 100–104, Nov 1994.

24. L. Willams. Pyramidal parametrics. *Computer Graphics*, 17(3), July 1983. ACM Siggraph '83 Conference Proceedings.

25. H. Yee, S. Pattanaik, and D. P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics*, 20(1):39–65, January 2001.

Appendix A: MATLAB Code

The MATLAB code given below implements the algorithm for computing threshold elevation for any given luminance image. To keep the code simple it has been assumed that the dimensions of the image are multiples of 8. The images that do not satisfy this requirement may be padded by reflecting or duplicating the boundary column or row pixel values. The 8x8 DCT matrix is derived from Equation 4 except that MATLAB uses indices from 1–8 rather than 0–7.

```
function c = thresholdmap(Y)
%
% Input:  Y - the Luminance im-
age. Range is 0-255.
%             Y = 0.299*R+0.587*G+0.114*B.
% Output: c - the threshold elevation map.
%

Q = GetJPEG_QuantizationMatrix;
T = Get8x8DCTmatrix;
T_t = T';
[rows,cols] = size(Y);
Y_dagger = Y;

for i = 1:8:rows
  for j = 1:8:cols
    F = T*Y(i:i+7,j:j+7)*T_t;
    absF = abs(F);
    Q_a = Q*(F(1,1)/1024);
    ind = find (absF >= Q_a);
    Q_m= Q_a.*max(1,(absF./Q_a).^0.7);
    Q_m(1,1) = Q_a(1,1);
    F_dagger = absF;
    F_dagger(ind) = F_dagger(ind)+Q_m(ind)/2;
    Y_dagger(i:i+7,j:j+7)
          = T_t*(sign(F).*F_dagger)*T;
  end;
end;

diff1 = abs(Y-Y_dagger);
diff2 = Y*(0.5*Q(1,1)/1024);
diff = max(diff1,diff2);
c = ones(rows,cols);
ind=find(diff2>0);
c(ind) = diff(ind)./diff2(ind);

return;
```