Enhancements to Directional Coherence Maps

Annette Scheel, Marc Stamminger, Joerg Puetz, Hans-Peter Seidel

Enhancements to Directional Coherence Maps

Annette Scheel
Marc Stamminger[1]
Jörg Pütz
Hans-Peter Seidel

Max-Planck-Institute for Computer Science
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
{stamminger,scheel,jpuetz,hpseidel}@mpi-sb.mpg.de

ABSTRACT

Directional Coherence Maps as proposed by Guo in '98 are a very efficient acceleration technique for ray tracing based global illumination renderers. It vastly reduces the number of pixels which have to be computed exactly by identifying regions which are suitable for interpolation. By using oriented finite elements for interpolation, the sampling density can be kept low for large regions of the target image. In this paper we describe extensions of the method. An improved object test is presented which prevents that small objects are missed. Additionally, it is shown how to handle textures efficiently, which was not possible with the original approach.

Keywords: Directional Coherence Maps, Rendering, Interpolation

1 Introduction

Creating a high-quality image from a global illumination solution can take hours for complex scenes if every pixel is evaluated with high accuracy. Therefore, several methods have been developed which decide which pixels can be approximated by interpolation.

An early, widely used strategy for adaptive refinement for ray tracing was developed by [Paint89]. They progressively add new image samples and store them in a k-D tree. The decision where to place new samples is based on the variance of the samples (to detect features) and on the area which is covered by them (to cover under-populated areas). They use a piecewise constant reconstruction from the cells of the k-D tree plus filtering. A large number of methods followed that tried to improve the strategies for finding discontinuities both in image and in object space and to exploit this knowledge for faster image synthesis. This includes for example methods for finding shadows [Chin89, Telle92, Drett94, Stewa94, Hart99], discontinuity meshing [Heckb92, Lisch93], subdividing image space according to object and shadow boundaries [Pighi97], and adaptive interpolation of indirect light [Ward92].

[1] currently at iMagis-GRAVIR/INRIA, Sophia-Antipolis, France

The Directional Coherence Maps of Guo [Guo98] provide an interpolation method even for finer details of the solution. This method tries to find the main orientation of shading details (a shadow or a glossy highlight) and object boundaries and interpolates along those discontinuities by using so-called oriented finite elements. In contrast to discontinuity meshing the performance of the method is not constrained by the number and type of lightsources and complex geometry calculations are avoided. The DCM reconstruction can be used in conjunction with every global illumination algorithm which is able to compute exact results for single pixels.

The time to compute a high-quality image is usually very long; therefore it is desirable to have already meaningful early images. With progressing computation the solution should improve and finally converge to the correct solution [Kajiy86]. The DCM incorporate such a progressive refinement as well by using a hierarchical block refinement technique.

In the following, first the DCM algorithm will be described in more detail (Section 2) and then our improvements for this method will be given (Section 3).

2 The Directional Coherence Maps

The goal of the Directional Coherence Maps by [Guo98] is to compute as few image points as possible exactly and to determine the other points by using some interpolation scheme.

Of course anti-aliasing is an issue for all ray-tracing based rendering systems. In order to keep things simple, we will assume in the following that anti-aliased images are obtained by creating images of multiple resolution and scaling them down to the desired image size. In the following a *pixel* is an image point of this enlarged high resolution image, several such pixels are averaged to obtained a single pixel value of the final image.

2.1 Overview

The image space is first subdivided into so called *elementary blocks* which are small squares, typically of size 8x8 pixels. At the beginning of the DCM reconstruction each elementary block is classified either as a *smooth block* or an *edge block* (see Figure 1 a) and b) and 2.2 for a description of the classification procedure).

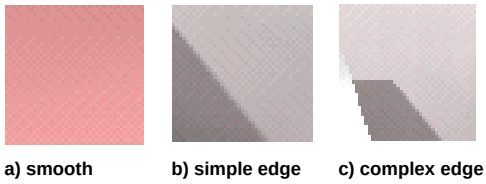

Figure 1: Block types

Smooth blocks are considered to have no discontinuities inside. For those blocks, only the four corners are evaluated exactly—all other pixels are determined by bilinear interpolation between those four corner values.

In contrast, edge blocks contain one or more discontinuities and simple bilinear interpolation would not be sufficient in this case. For those blocks all pixels on the block's border are evaluated exactly. With this information the direction of the "strongest" discontinuity is found. This is done by determining the least discrepancy direction, i.e. the direction for which the differences between border pixels that are opposite with respect to this direction are minimal (see Figure 2). Typically, eight different directions are tested. Parallel to this discontinuity so called *oriented finite elements* are established, which are longish quadratic polygons (see Figure 2). The interior of those elements is obtained by linear interpolation between the border values.

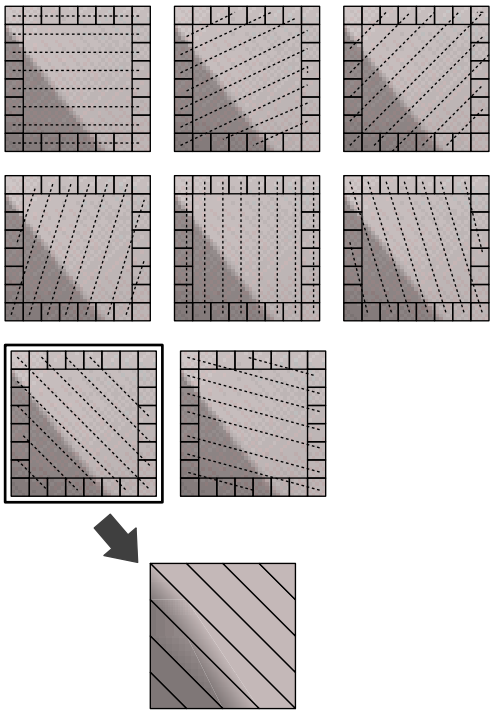

Figure 2: Finding the least discrepancy direction and drawing the oriented finite elements

If the desired quality is not yet reached—i.e. important features have been missed—the DCM reconstruction can be iterated by subdividing every block into four quads and classifying the children into smooth or edge blocks, again. If finally blocks have the size of one pixel the result is the same as a baseline renderer would produce.

In order not to miss any features, the four children of a smooth block undergo the same classification into smooth or edge blocks as elementary blocks (see 2.2). For edge blocks another test (see 2.3) is done before they are subdivided into four quads in order to determine if they contain only one single discontinuity which is only slightly curved (so-called *simple edge* blocks) or some more complex features (*complex edge* blocks) (see Figure 1 b) and c)). For children of complex edge blocks the oriented finite elements have to be re-calculated, whereas for simple edge blocks only a *lazy* boundary evaluation is done. Here, only the center pixel (of the parent) is computed exactly and this value is compared to the result of the finite element approximation. If this does not differ significantly the old orientation of the finite elements is kept, the interpolation is just corrected with the new values. Otherwise, a new full boundary evaluation has to be done.

2.2 Classification Smooth—Edge Block

Two tests are done to determine if a block is a smooth or an edge block. The first test is a contrast test: From the exact radiance values at the four vertices the maximum and minimum luminances are determined to compute the block contrast following Mitchells definition of contrast: $\frac{max-min}{max+min}$. If this contrast is above some threshold, the block is classified as an edge block. For the other test, all visible lines from object boundaries (as well as their reflection in planar mirrors) are computed by creating an object-ID index image with OpenGL. If a block is crossed by a visible line, it is classified as an edge block, too.

2.3 Classification Simple—Complex Edge Block

A block has to pass three tests before it can be confirmed to be a simple edge block — otherwise it will be classified as a complex edge block. The first test (the so-called *zero order* test) tries to find the number of edges crossing this block by counting the significant changes in color along the border. If there are only two changes, it can be supposed that there is only one discontinuity (see Figure 3).

The next test, the *first order* test, attempts to find the gradient of the discontinuity for each of the two points, where the discontinuity intersects the block border. For this purpose, the pixels around the crossing point inside the border have to be evaluated. If the gradient is nearly the same at both crossing points, the discontinuity (probably) is a straight line (see Figure 3).

The last test uses the object IDs which were already used for the smooth—edge block classification. No more than two objects are allowed on the block border for a simple edge block.

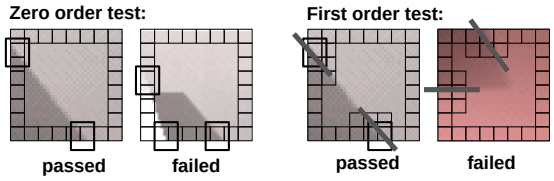

Figure 3: Zero and first order tests

3 Enhancements of the Algorithm

In our implementation of the algorithm of Guo we achieved similar results as were described in the original paper (see Figures 4 and 7). Depending on the scene's character only 6–10% of the pixels had to be evaluated exactly without a noticeable loss of quality. Most images were produced with oversampling (4 or 16 times) to obtain anti-aliased results. For large images (i.e. screen size) it might be possible that the data used for the DCM does not fit into main memory. We therefore implemented a possibility to subdivide the image into smaller parts and to do the algorithm for all sub-images independently.

The remainder of this section describes enhancements we introduced in order to handle thin objects on the finest block refinement level (see 3.1) and to improve the object tests which are used in several occasions during the algorithm (see 3.2). Furthermore, because the original algorithm was not able to handle textures efficiently, we developed an enhancement for textures which will be described in 3.3.

3.1 Handling Thin Objects

The DCM algorithm is designed to create progressively finer and better images, theoretically starting with a single elementary block containing the entire image and finally reaching pixel level. In practice it is a good starting point to use elementary blocks of size 8x8 (corresponding to Guo and our experience, too). In our implementation we then refine only once to obtain 4x4 blocks. For the next finer block size of 2x2 it would not make sense to do the complex finite element orientation procedure.

Especially very thin and longish objects like the shaft of the desk lamp of Figure 4 or the pricks of the cactus in Figure 7 which only have a thickness of 1 or 2 pixels are difficult to represent by the finite element interpolation. Blocks containing those objects will be classified as (complex) edge blocks due to the object ID tests. The finite element orientation procedure tries then to find the least discrepancy direction. Because the thin line only covers very few border pixels it will hardly contribute to overall differences. Especially in the presence of other shading details (like shadow) having a different orientation than the line the finite elements will not be oriented into the direction of the thin line. Therefore, the thin line will be washed out by the interpolation with this scheme (cp. Figure 5, left image).

We therefore use another test after the finite elements of a 4x4 block have been established. Before a value of an interior pixel is determined by interpolation it is tested if the object IDs of the border pixels correspond to the object ID of this pixel (the object IDs are already known from the classification tests 2.2). If not, this single pixel is computed exactly.

The images of Figures 4 and 7 were computed with this extension. The right image of Figure 5 contains a detail of Figure 7. Especially the pricks of the cactus are reproduced better than in the left image which was created by the original model. The index image

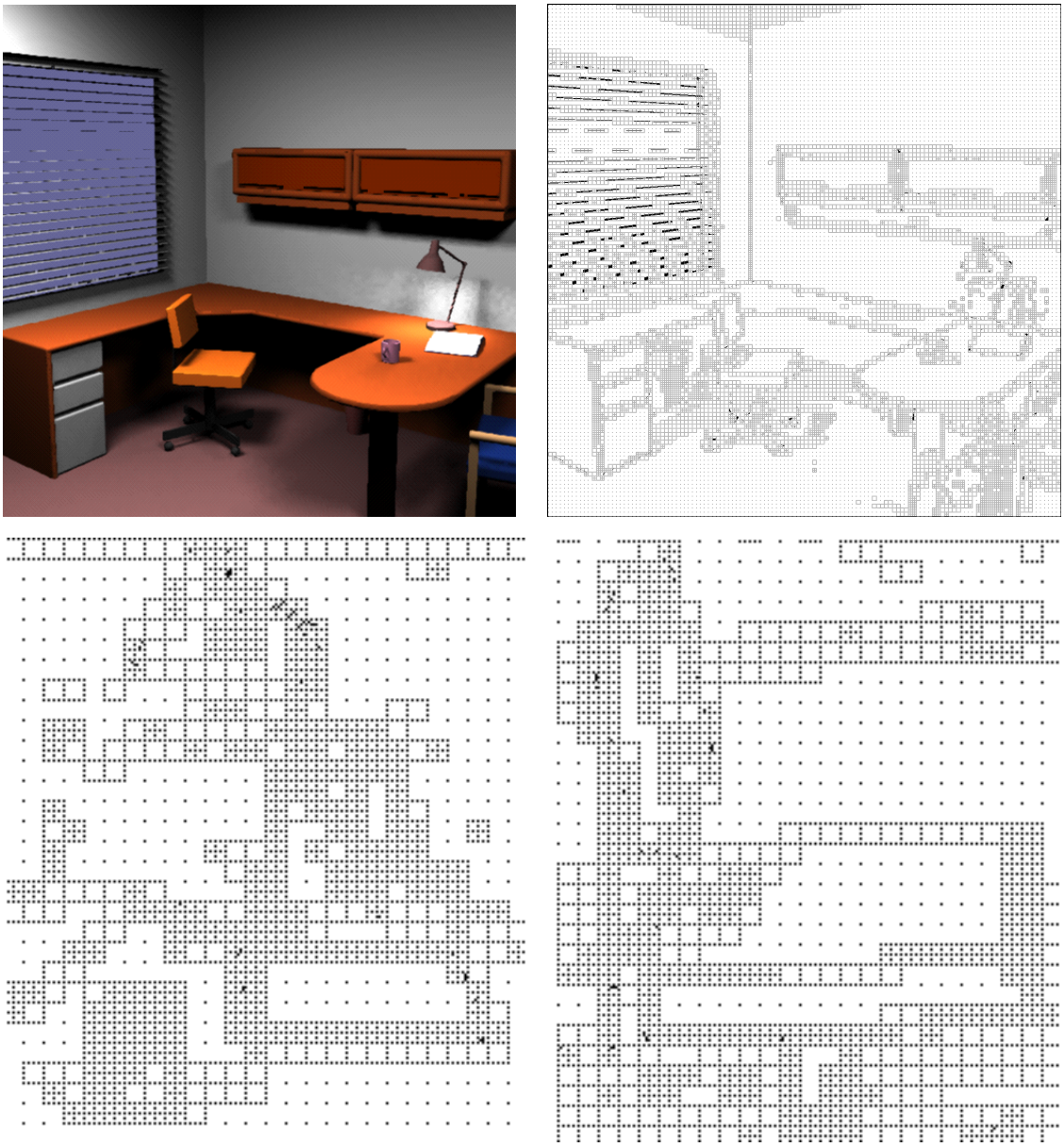

Figure 4: Size 512x512, 4 times oversampling, 6% of pixels are computed exactly. The bottom images show two details of the index image (desk lamp and chair).

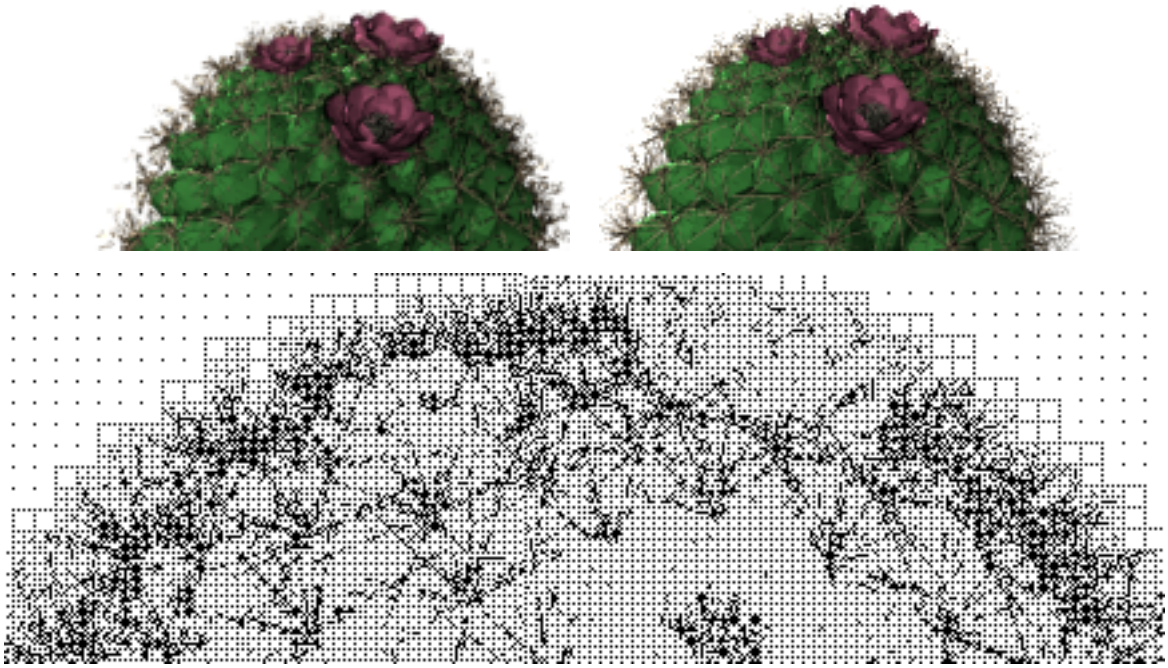

Figure 5: Detail of the cactus. Left: original method (refined to 4x4 blocks), Right: new method, Bottom: Enlarged part of the index for the new method.

on the bottom shows which pixels are computed additionally (those not having a block structure). The total increase in exactly computed samples was only about 1% for this scene.

3.2 Improved Object Test

The original algorithm uses object IDs for the smooth—edge block as well for the simple—complex edge block classification. However, often more complex objects are composed from simple drawing primitives like triangles all consisting of the same material. In this case, it is desirable to interpolate across those primitives in order not to stress the borders between them. Therefore, we additionally compare the shaders of the objects. Only if object ID **and** shader differ, blocks will be classified as edge or complex edge blocks, respectively. Because shaders can be of arbitrary complexity, we decided to compare only shader IDs.

3.3 Enhancement for Textures

Texturing is a very efficient and widely used means for increasing the visual richness of synthetic renderings. By mapping images onto geometric objects, even simple objects can offer high visual detail in the final rendering, almost without increase of overall rendering time.

Such high visual detail from textures, however, retards the method of Guo. The method considers the texture as lighting detail, which entails fine sampling of the entire object. Compare Figure 6, where the floor is a single polygon with a parquet texture. It exhibits fine detail both due to the parquet texture and due to lighting, in this case mainly the chair's shadow.

However, texture detail is usually produced very lately, namely at the last light bounce before the light reaches the eye (unless you look onto textured surfaces via a mirror for example). It is not the incident light at the object that changes rapidly, but just its reflection.

This distinction opens the way for improving Guo's method in the presence of textured objects: if we could interpolate the *incident* light at the object with Guo's method and perform the reflection computation including the textures afterwards, in the above example the method can well interpolate large regions of the floor and can concentrate on the critical shadow boundaries and the glossy reflection. The right column of Figure 6 shows the result of our approach based on this idea. Although the final image has rich detail allover due to the textured floor, only the shadow boundaries and glossy reflection boundaries were sampled finely. In contrast, the left column of Figure 6 shows that the textured floor is sampled densely when the original approach is used.

This approach was chosen by [Pighi97], but interpolating incident light is only possible if the last reflection is diffuse. To be able to handle more complex

BRDFs we only delay the texturing and thus the most rapidly changing part of reflection until the final interpolation. More precisely, we split the reflected light L of a pixel into two parts: the first part L_t is the one that has been modulated by the object's texture, the second one L_c is the constant offset unaffected by the texture. Note that e.g. for plastic a texture only modulates the diffuse reflection of an object ($\rightarrow L_t$), whereas the glossy reflection does not vary ($\rightarrow L_c$).

Instead of L_t, we consider the value *before* texturing, i.e. the value $L_i = L_t/\tau(u,v)$. Altogether, if we have an image sample on an object at texture coordinates (u,v), and the texture function on the object is $\tau(u,v)$, the resulting color of the pixel is

$$L = L_t + L_c = L_i\tau(u,v) + L_c,$$

The gain of this separation is that we now have two colors L_i and L_c that are about as smooth as if the scene was not textured at all. If we apply Guo's approach to this color tuple (L_i, L_c) instead of to L, less samples will become necessary.

On the downside, the interpolation becomes more complicated. First, we have to consider two colors, i.e. six color channels, for interpolation, contrast computation etc. Second, we have to evaluate the texture function $\tau(u,v)$ during interpolation, which means that we have to compute the texture coordinates (u,v) for an interpolated point and have to evaluate the texture to obtain the modulation factor.

Six color channels instead of three is just a small memory problem. More difficult is the computation of texture coordinates. But since we know from the tests using object IDs for each interpolation point which object it sees, we can quickly intersect the corresponding eye ray with the object to obtain the texture coordinates.

4 Results and Conclusion

The image reconstruction method of Guo turned out to be a very efficient means for the acceleration of ray-tracing based renderers. Image discontinuities are detected reliably, and with the oriented finite elements simple discontinuities are approximated very efficiently. In particular if supersampling is used for anti-aliasing and the final images are obtained by averaging the reconstruction result, the quality loss is low.

In the following table we compared the method of Guo with a simple baseline renderer for the scene in Figure 4. The computation time was split into the time needed for pure illumination computation and for the DCM. The table shows that the time overhead introduced by the DCM is rather small but the overall time gain is high, although we used only a very simple illumination computation algorithm (300 rays are shot to the lightsources, for glossy objects 20 reflection rays are shot additionally). For a more sophisticated illumination algorithm (like for example a final gather of a finite element solution) even a bigger time gain can be expected.

Baseline vs. DCM (for Figure 4)			
Method	Pixel	Time DCM	Time Illum
Baseline	100 %	–	3672 s
DCM	5.9 %	43 s	245 s

With our improved testing scheme, most complex cases where the original method fails are detected. By switching back to a full sampling for these few critical cases many small artifacts can be avoided by only a slight decrease of rendering performance. As a result, the quality of the final images, although obtained from usually less than 10% of the samples, is close to that of a fully sampled image (see Figure 7).

The table below contains a comparison of Guo's original method (refined to 4x4) and the extension for thin objects for the scene in Figure 7. The higher quality of the image was achieved by the expense of only approximately 1% more samples. The time for the DCM computation is moderately increased because of the additional test during the interpolation. The scene is only directly lit by point light sources, therefore the DCM time exceeds the time for the illumination computation. Again, with a more complex lighting algorithm the time for illumination computation will dominate over the time needed for the DCM.

Thin Objects (see Figure 7)			
Method	Pixel	Time DCM	Time Illum
Guo	7.28 %	70.82 s	52.38 s
New	8.17 %	94.16 s	55.42 s

Furthermore, we could solve the problem of very dense sampling enforced by textured surfaces. By delaying the texturing to the final interpolation step, textured surfaces are handled without an increase in sampling density (see Figures 6 and 7).

In the following table we compared the improved method with Guo's original method for the scene in Figure 6. It shows that the number of exactly computed samples and hence also the computation time could be strongly reduced, while the increase in the time needed for the DCM computation is minimal. Again, with a more complex lighting algorithm an even higher time gain can be expected.

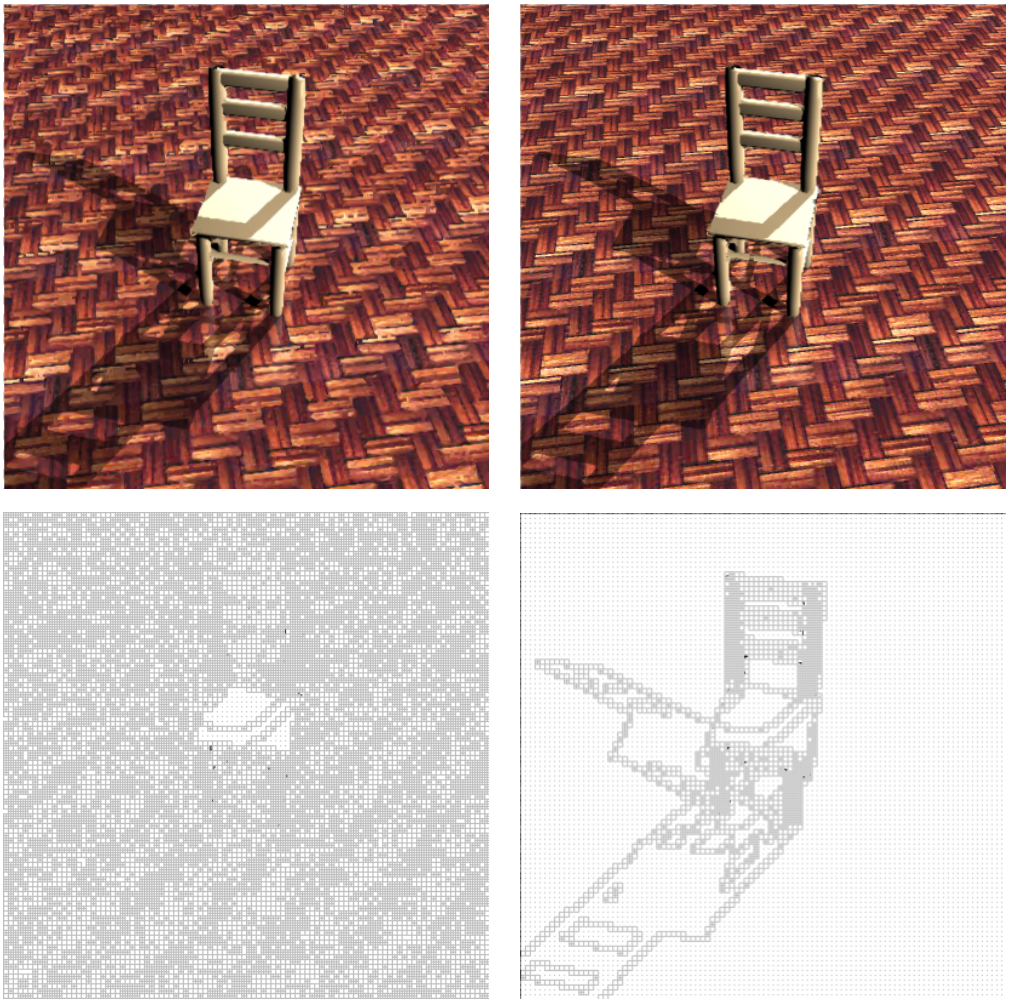

Figure 6: Left: without texture extension, Right: with texture extension

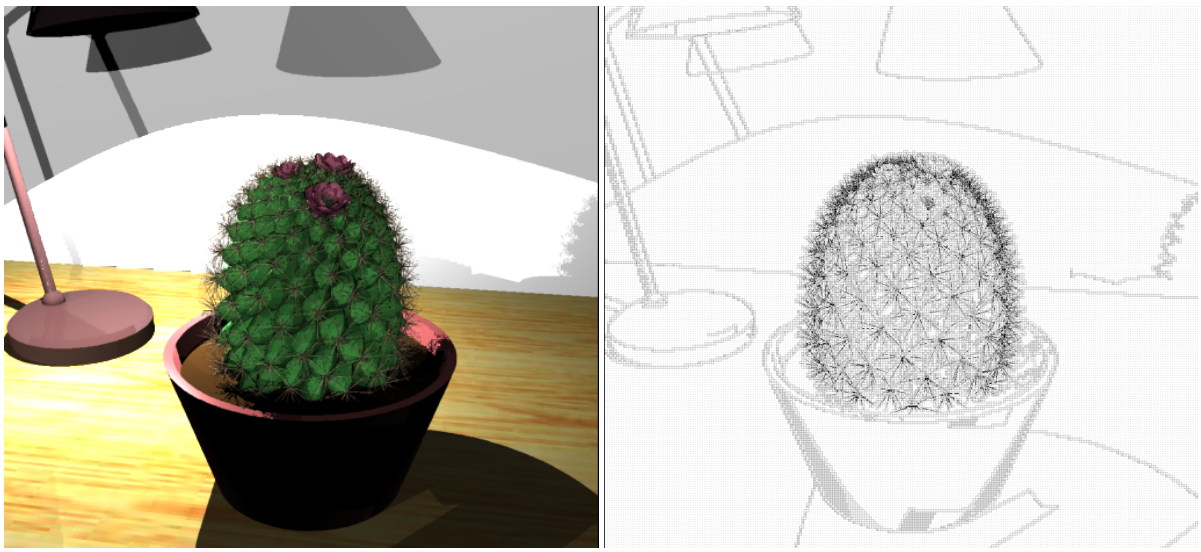

Figure 7: Size 512x512, 16 times oversampling, 7% of pixels computed exactly. The cactus model was created by Oliver Deussen and Bernd Lintermann.

Textures (see Figure 6)			
Method	Pixel	Time DCM	Time Illum
Guo	17.8 %	40.66 s	27.93 s
New	3.6 %	42.64 s	7.08 s

On the downside, the method suffers from noisy illumination computations. The noise cannot be distinguished from 'real' detail and results in too fine sampling of the noisy regions. We are not aware of a simple solution to this problem.

5 Acknowledgments

This research was funded by the ESPRIT Open LTR Project #35772 SIMULGEN "Simulation of Light for General Environments".

REFERENCES

[Chin89] N. Chin and S. Feiner. Near real-time shadow generation using bsp trees. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, pages 99–106, July 1989.

[Drett94] George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. In *Computer Graphics (SIGGRAPH'94 Proceedings)*, pages 223–230, July 1994.

[Guo98] Baining Guo. Progressive radiance evaluation using directional coherence maps. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 255–266, 1998.

[Hart99] David Hart, Philip Dutre, and Donald Greenberg. Direct illumination with lazy visibility evaluation. In *Computer Graphics (SIGGRAPH'99 Proceedings)*, pages 147–155, August 1999.

[Heckb92] Paul S. Heckbert. Discontinuity meshing for radiosity. In *Third Eurographics Workshop on Rendering*, pages 203–226, May 1992.

[Kajiy86] James T. Kajiya. The rendering equation. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986.

[Lisch93] Daniel Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 199–208, 1993.

[Paint89] James Painter and Kenneth Sloan. Antialiased ray tracing by adaptive progressive refinement. In *Computer Graphics (Proceedings of SIGGRAPH '89)*, volume 23, pages 281–288, 1989.

[Pighi97] F. Pighin, D. Lischinski, and D. Salesin. Progressive previewing of ray-traced images using image-plane discontinuity meshing. In *Eurographics Workshop on Rendering 1997*, May 1997.

[Stewa94] A. J. Stewart and S. Ghali. Fast computation of shadow boundaries using spatial coherence and backprojections. In *Computer Graphics (SIGGRAPH'94 Proceedings)*, pages 231–238, July 1994.

[Telle92] S. Teller. Computing the antipenumbra of an area light source. In *Computer Graphics (SIGGRAPH'92 Proceedings)*, pages 139–148, July 1992.

[Ward92] Gregory J. Ward and Paul Heckbert. Irradiance gradients. *Third Eurographics Workshop on Rendering*, pages 85–98, May 1992.