11. David Salesin, Dani Lischinski, and Tony DeRose. Reconstructing illumination functions with selected discontinuities. In *Proceedings of the Third Eurographics Workshop on Rendering (Bristol, UK, May 18–20, 1992)*, pages 99–112, May 1992.
12. Mike Salisbury, Corin Anderson, Dani Lischinski, and David H. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, pages 461–468, August 1996.
13. Carlo H. Sequin and Eliot K. Smyrl. Parametrized ray-tracing. In *Proceedings of SIGGRAPH '89 (Boston, Massachusetts, July 31–August 4, 1989)*, volume 23, pages 307–314, July 1989.
14. Seth Teller, Kavita Bala, and Julie Dorsey. Conservative radiance interpolants for ray tracing. In *Proceedings of the Seventh Eurographics Workshop on Rendering (Porto, Portugal, June 1996)*, June 1996.
15. Greg Ward. The RADIANCE lighting simulation and rendering system. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 459–472, July 1994.
16. Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of SIGGRAPH '88 (Atlanta, Georgia, August 1–5, 1988)*, volume 22, pages 85–92, August 1988.
17. Georges Winkenbach and David H. Salesin. Rendering parametric surfaces in pen and ink. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, pages 469–476, August 1996.

edges reflected on curved surfaces. To handle soft shadows we could use an extension of the shadow volume algorithm to area light sources [3].

We could use our method to compute ray-traced walkthroughs, exploiting frame coherence for reusing samples between successive frames. Determining which samples could be reused would depend on the movement of the viewer and the materials of the scene. We believe that our method could be very effective for this task if the scene contained many Lambertian surfaces.

6 Acknowledgements

This work was supported in part by an Alfred P. Sloan Research Fellowship (BR-3495), an NSF Presidential Faculty Fellow award (CCR-9553199) and Postdoctoral Research Associates Award (CDA-9404959), an ONR Young Investigator award (N00014-95-1-0728) and Augementation award (N00014-90-J-P00002), and an industrial gift from Microsoft.

References

1. Normand Brière and Pierre Poulin. Hierarchical view-dependent structures for interactive scene manipulation. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, pages 83–90, August 1996.
2. Norman Chin and Steven Feiner. Near real-time shadow generation using bsp trees. In *Proceedings of SIGGRAPH '89 (Boston, Massachusetts, July 31–August 4, 1989)*, volume 23, pages 99–106, July 1989.
3. Norman Chin and Steven Feiner. Fast object-precision shadow generation for area light sources using bsp trees. In *Proceedings of 1992 Symposium on Interactive 3D Graphics (Cambridge, Massachusetts, March 29–April 1, 1992)*, March 1992.
4. Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.
5. Pat Hanrahan. Using caching and breadth-first search to speed up ray tracing. In *Proceedings of Graphics Interface '86 (Vancouver, British Columbia, May 26–30, 1986)*, pages 56–61, May 1986.
6. Paul S. Heckbert. Discontinuity meshing for radiosity. In *Proceedings of the Third Eurographics Workshop on Rendering (Bristol, UK, May 18–20, 1992)*, pages 203–216, May 1992.
7. Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, November 1992.
8. Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In *Proceedings of SIGGRAPH '93 (Anaheim, California, August 1–6, 1993)*, pages 199–208, August 1993.
9. Dani Lischinski. Incremental Delaunay triangulation. In *Graphics Gems IV*, Paul S. Heckbert, editor, Academic Press, 1994.
10. James Painter and Kenneth Sloan. Antialiased ray tracing by adaptive progressive refinement. In *Proceedings of SIGGRAPH '89 (Boston, Massachusetts, July 31–August 4, 1989)*, volume 23, pages 281–288. ACM, July 1989.

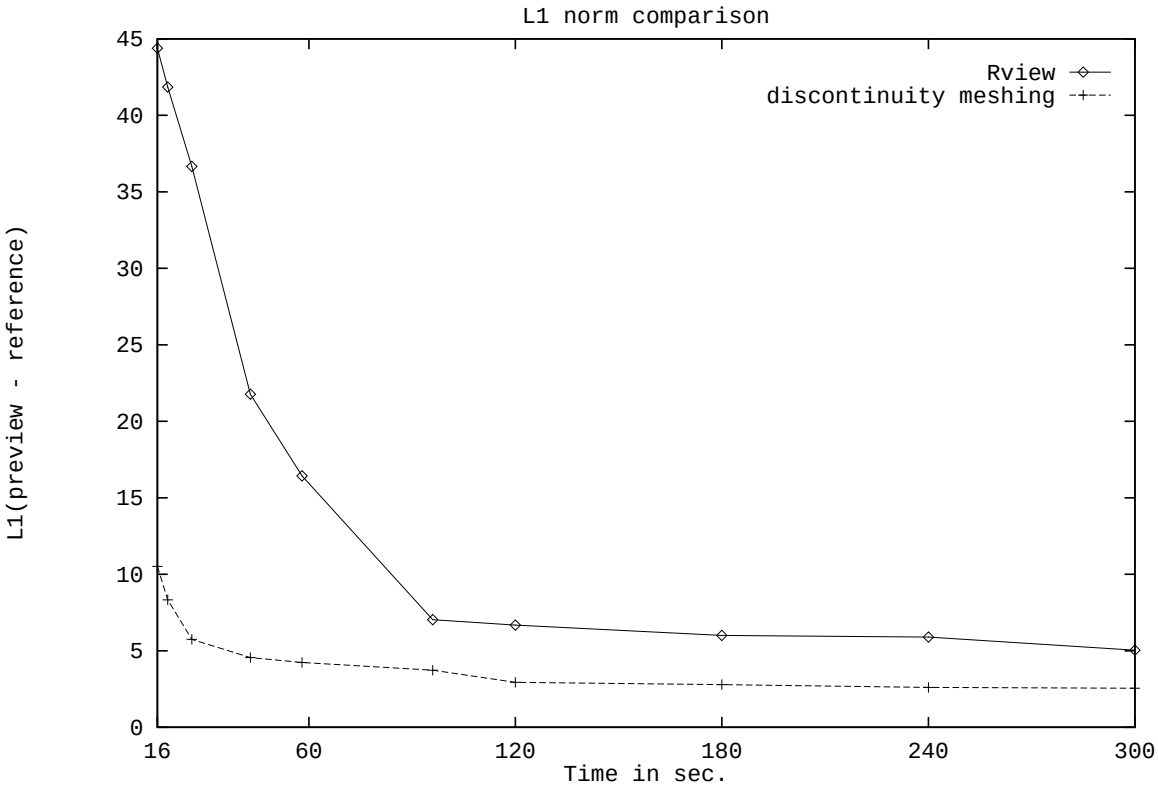


Fig. 4. *Preview error in L_1 norm.*

5 Conclusion

We have described a new method for progressively previewing ray-traced images. Our initial implementation compares favorably with other previewers, yielding visually accurate approximations to the final image early on. An interesting observation is that the computation of the preview is almost resolution independent. The only part that depends on the resolution is the processing of the procedural texture maps. Because discontinuities in the image are explicitly represented as constrained edges in the triangulation, magnifying the image does not result in much blurring, similarly to the magnification algorithm described by Salisbury *et al.* [12].

The most important limitation of our approach is that it is currently capable of handling polygonal scenes only. To extend our previewer to handle curved objects, we could either tessellate those objects and use the same BSP-tree-based shadow volume algorithm, or use a different algorithm, such as the one used by Winkenbach and Salesin [17] since it does not rely on BSP-trees.

Another possible extension would be to include more types of discontinuity edges in out algorithm. For instance, we could compute soft shadow edges, refracted edges, and

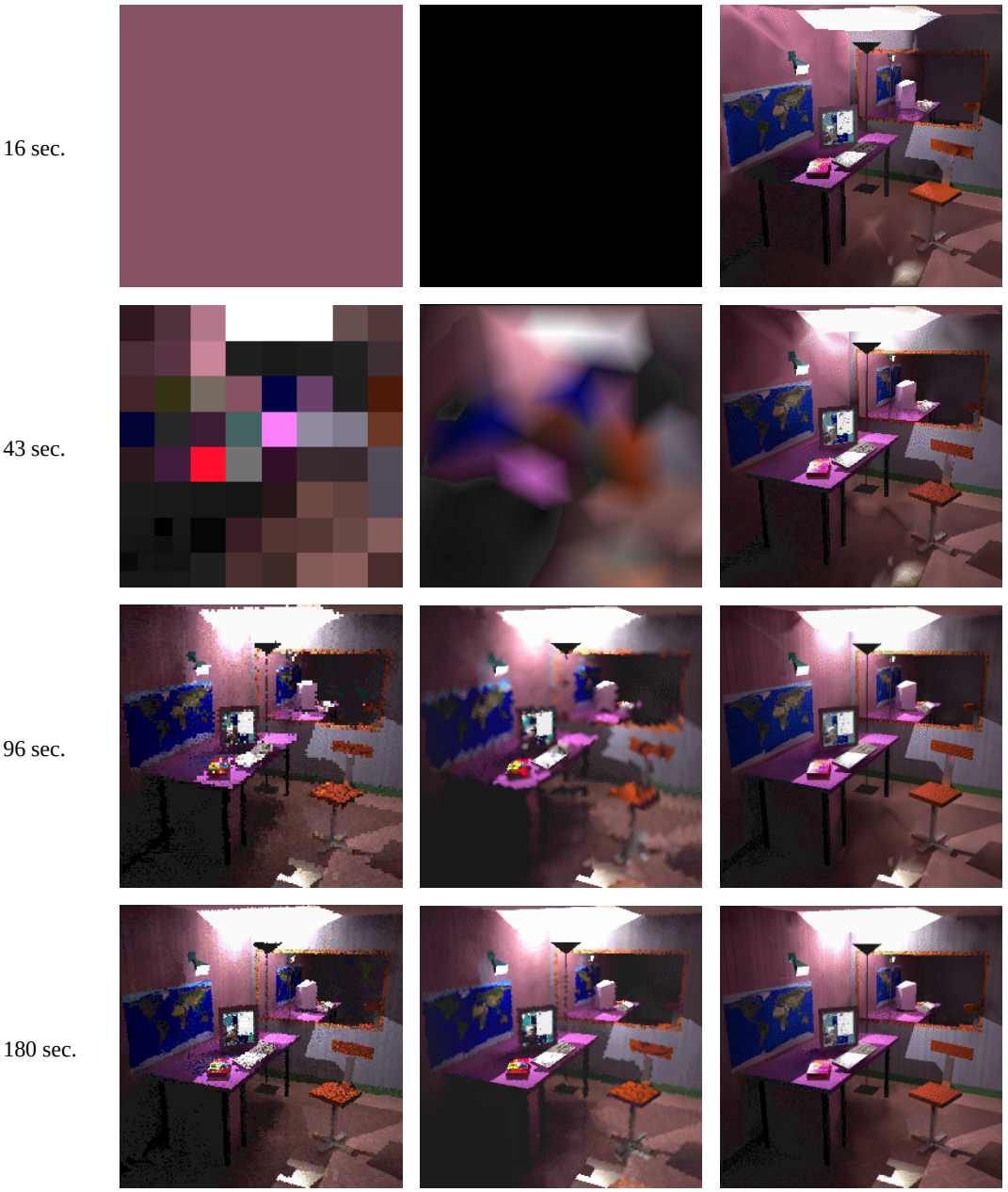


Fig. 3. *A comparison between three different previewing techniques.*

3 Computation of constrained edges

Our previewer requires the computation of the visible geometric edges, the visible shadow edges, and the visible reflections of these edges in order to insert their image-plane projections into the constrained triangulation. Our current implementation is limited to polyhedral scenes illuminated with point light sources, and it repeatedly uses the BSP-tree-based *shadow volume algorithm* (SVA) proposed by Chin and Feiner [2] to compute all of these edges.

To determine which geometric edges are visible in the image, we use the SVA to construct a shadow volume defined by the eye and all of the scene polygons that are contained in the view frustum. Any scene edge, or part thereof, that is not contained in this shadow volume is a visible edge.

Next, we process the light sources in the scene. For each light source, we construct a shadow volume and use it to find all of the shadow edges in object space. Each shadow edge is tested against the eye's shadow volume to determine whether or not it is visible.

Finally, we process the mirrors in the scene. For each visible mirror, we compute the reflected location of the eye with respect to the mirror's plane. The resulting virtual viewpoint is used to construct yet another shadow volume. This time the "shadowing" polygons must be contained in the view frustum defined by the eye and the mirror polygon. Each geometric or shadow edge visible from the virtual viewpoint is first projected onto the mirror, and then tested against the eye's shadow volume to determine whether or not it is visible in the image. Note that we only account for one level of reflection. Recursive reflections could be handled in a similar way, but would result in an exponential growth in the computation.

4 Results

This section demonstrates the performance of our previewer on a sample office scene. The experiment was conducted on an Indigo2 Maximum Impact with a 200 MHz R4400 processor and 128 megabytes of RAM.

Figure 3 shows three columns of images of the test scene that were displayed by three different previewers at four different points in time. The images in the left column were generated using RADIANCE's native previewer *Rview*, which progressively increases the resolution of the image. The images in the middle column were generated by a technique similar to the one suggested by Painter and Sloan [10]: adaptively computed image samples are inserted into a plain Delaunay triangulation, which is displayed using hardware Gouraud shading. The images in the right column were computed using our previewer. All three previewers are using RADIANCE's ray-tracing engine with high accuracy requirements for the indirect component.

For this scene, it takes our previewer one second to compute the discontinuity edges and to construct a constrained Delaunay triangulation containing them. After sixteen seconds of computation our previewer already displays a perfectly usable approximation to the final image, while the other two previewers generate clearly inferior results (blocky and blurry, respectively) for the first two minutes of computation. As time goes by, the difference between the three previews gradually diminishes; however, our preview remains superior in quality. These results are illustrated quantitatively in Figure 4, which shows a plot of the L_1-norm differences between images generated by *Rview* and by our previewer and the completely ray-traced image.

1. Compute (in object space) the set E, which is the union of all visible geometric edges, all visible shadow edges, and all visible mirror reflections of geometric and shadow edges. (This step is described in more detail in the next section.)
2. Construct a constrained Delaunay triangulation using a sparse regular grid of points. Insert into the triangulation the projections of the edges in E as constrained edges (image (b)).
3. Render in hardware (without illumination) all surfaces that are texture mapped with images, and save the result in the *DetailMap* (image (c)).
4. Sample the areas of the image containing procedurally textured surfaces. To determine which pixels must be sampled we render those surfaces with unique colors (image (d)). At each such pixel, compute the color of the procedural texture and store the result in the *DetailMap* (image (e)).
5. Produce the initial preview image by tracing rays through the vertices of the constrained triangulation and displaying the result with Gouraud shading (using the *DetailMap* for texture mapping, where necessary). See image (f).
6. Progressively refine the triangulation until the user stops the previewing, or until the image has been completely sampled. Image (h) shows a preview generated after further refinement of the triangulation. Note the BF-squares that have appeared in the *DetailMap* (image (g)).

Our experience with the above algorithm has shown that sampling all the procedurally textured surfaces ahead of time takes too long if those surfaces cover a large proportion of the image plane. Therefore, we decided to perform this sampling only when and where needed, while refining the triangulation. We sample the procedural texture on a particular surface only when the sample density over its projection exceeds a certain threshold. This improvement spreads the cost of sampling the procedural textures over the course of the preview. Moreover, in this way we avoid sampling procedurally textured objects that are not sufficiently illuminated.

Implementation

We chose the RADIANCE lighting simulation and rendering system [15] as a testbed for out previewing algorithm. Our algorithm uses RADIANCE's ray-tracing engine to evaluate the image function at the sample points that are used to construct the triangulation of the image plane. Thus, we have in effect implemented an alternative previewer for RADIANCE. RADIANCE's native previewer, *Rview*, was used for performing comparisons with our method, as described in section 4.

Unlike ordinary ray tracers, RADIANCE employs a sophisticated algorithm for computing the indirect component of illumination resulting from diffuse inter-reflections. This computation is expensive, so its results are cached at various locations in the scene for reusal by subsequent nearby samples [16]. RADIANCE provides various parameters to control the accuracy of its rendering, in general, and of the indirect component, in particular. Naturally, if the desired accuracy is high, even the first preview takes a long time to compute. In order to provide the first preview quickly, our implementation starts by casting rays with low accuracy requirements. Gradually, the accuracy is increased and the image samples are recomputed until the target accuracy is reached. Further samples are computed directly at the desired accuracy, since they can take advantage of RADIANCE's cache, which has been warmed up in the meantime.

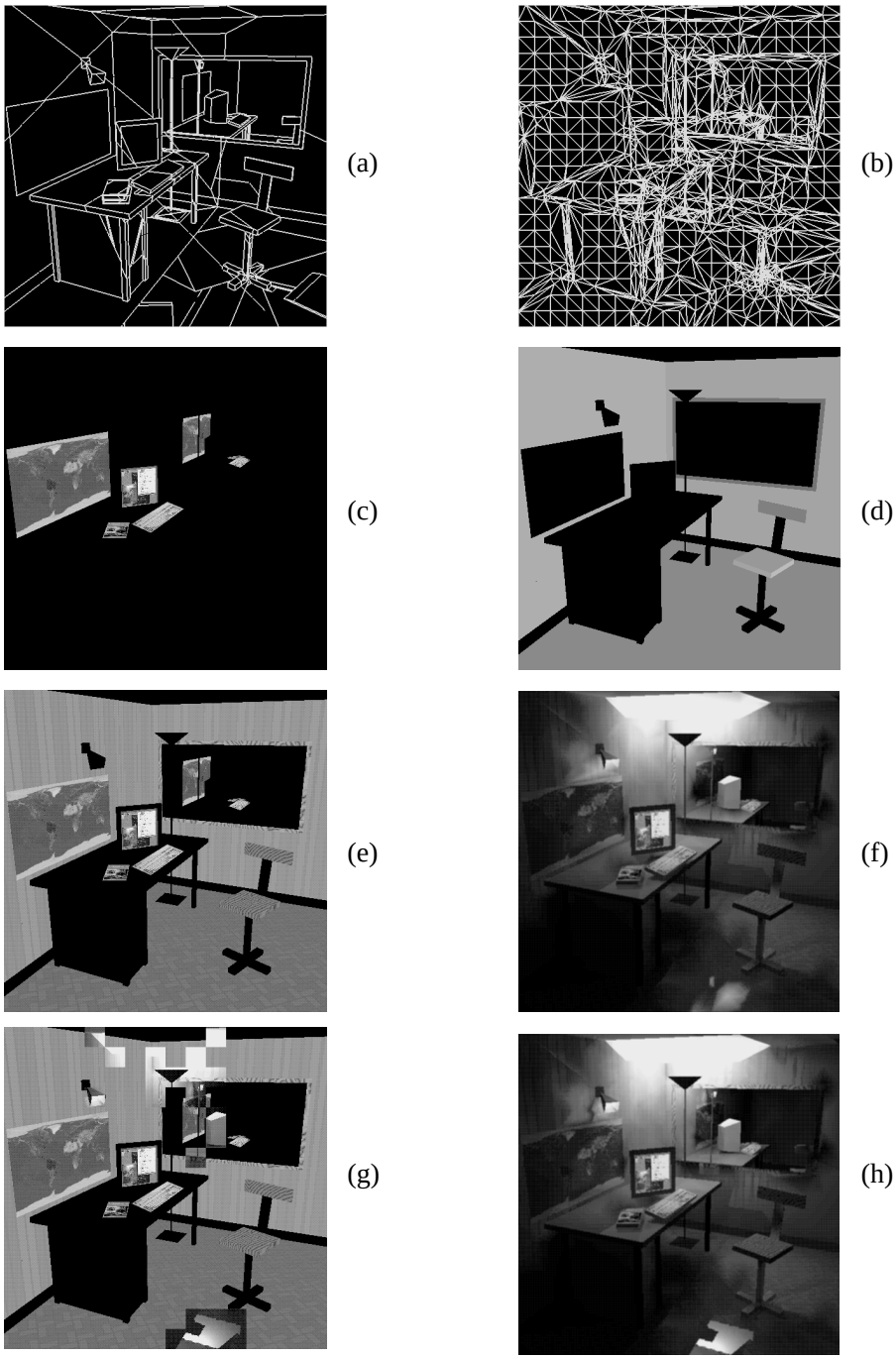


Fig. 2. *Phases of the previewing algorithm.*

this problem, we assign a weight to each edge in the triangulation and place the edges in a heap data structure. The triangulation is repeatedly refined by inserting a vertex in the middle of the "heaviest" edge in the heap. Our current implementation uses a simple weight function geared towards providing good coverage of the entire image area, as well as refining regions that exhibit substantial variations in color. Given an edge (v_1, v_2) in the triangulation, the weight $W(v_1, v_2)$ is given by:

$$W(v_1, v_2) = ||v_1 - v_2||_2 + ||v_1 - v_2||_{\text{color}}$$

where

- $||v_1 - v_2||_2$ is simply the Euclidean length of the edge (vertex coordinates are normalized to lie in $[0, 1]^2$),
- $||v_1 - v_2||_{\text{color}} = 0.3(v_1^{\text{red}} - v_2^{\text{red}})^2 + 0.59(v_1^{\text{green}} - v_2^{\text{green}})^2 + 0.11(v_1^{\text{blue}} - v_2^{\text{blue}})^2$, with the colors expressed as floating point values in $[0, 1]$.

As noted earlier in the paper, relying on adaptive refinement to reconstruct small details of the image can require a very large number of samples in regions of the image that exhibit high frequencies. When the number of samples becomes too large, the triangulation becomes expensive to manage. In such regions piecewise-linear interpolation of the image function is simply ineffective. Instead, we utilize hardware texture mapping as explained below.

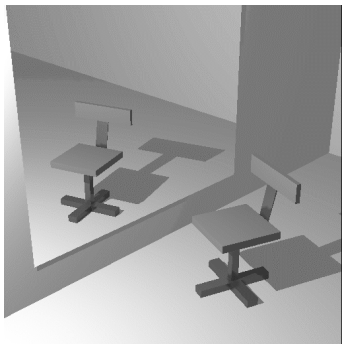
Most high-frequency regions in the image correspond to texture-mapped surfaces in the scene. Such regions should not require a high density of image samples if the incident illumination on those surfaces is relatively smooth and thus can be well approximated by linear interpolation. Our approach is then to render the texture-mapped surfaces in a pre-processing step, without applying illumination, and store the result. When an image sample falls on a textured region in the image, we store the illumination incident at that sample point without multiplying it by the corresponding texture-map reflectance value. To display a triangle covering a textured region, the illumination values stored at the vertices are linearly interpolated and the result is blended with the texture map value at each pixel. The interpolation and the blending are performed entirely by the Gouraud-shading and texture-mapping hardware.

To detect other (non texture-mapped) regions in the image that exhibit high frequencies, we superimpose a regular grid over the image plane. When the density of samples in a given square goes beyond a certain threshold we sample every pixel in the square and stop refining this area (we refer to such sampled squares as *brute-force squares* or *BF-squares* for short). Triangles that cover BF-squares are displayed using the BF-square as a texture map.

To implement the two schemes just described we maintain a separate image that contains both the texture mapped areas rendered without illumination and the BF-squares. This image is referred to as the *DetailMap*.

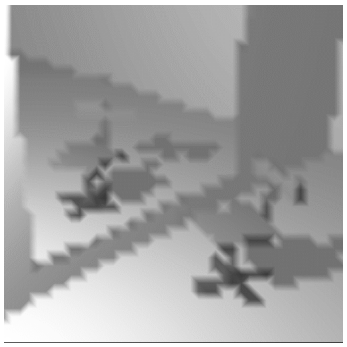
Our algorithm is restated in detail below, using Figure 2 to illustrate its various components:

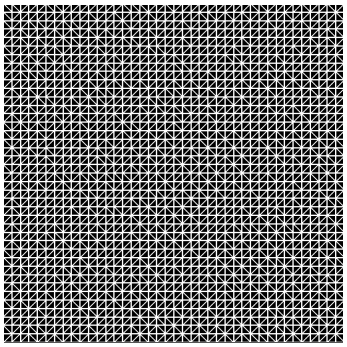
Raytracing



(a)

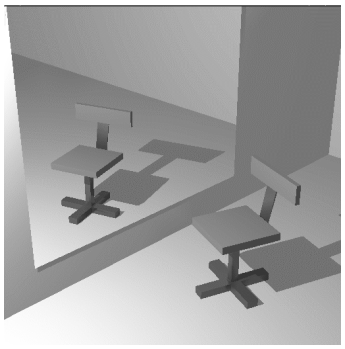
Uniform sampling

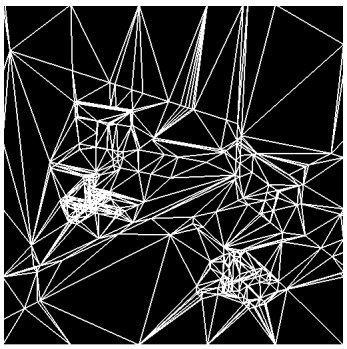


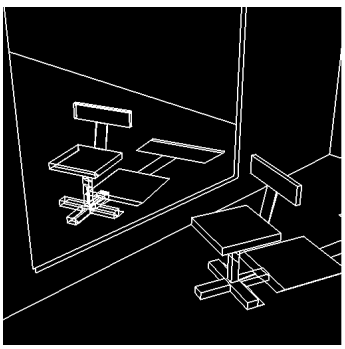


(b) Gouraud shading (c) Triangulation

Constrained triangulation







(d) Gouraud shading (e) Triangulation (f) Constrained edges

Fig. 1. *Comparison between regular meshing and discontinuity meshing.*

Another promising interpolation-based approach is described by Teller *et al.* [14]. This approach lazily constructs radiance interpolants in ray space where the ray trees have the same topology.

Discontinuity Meshing: Discontinuity meshing is the idea of explicitly representing function discontinuities in a mesh used to construct an approximation to the function. Heckbert [6] and Lischinski *et al.* [7, 8, 11] have used discontinuity meshing to drastically improve the accuracy of radiosity simulations. Salisbury *et al.* [12] used a discontinuity mesh containing sharp edges in an image in order to maintain the sharpness of these edges when the image is magnified.

2 Algorithm

The goal of our previewer is to generate an approximate image from a partial sparse set of image samples that is as close as possible to the final, completely sampled, ray-traced image. Furthermore, in order to provide a progressive preview, the previewer must update the approximation quickly as more image samples are computed. As originally suggested by Painter and Sloan [10], one possible approach is to triangulate the image plane using a Delaunay triangulation of the computed image samples. Since each vertex in the triangulation is associated with the color of an image sample, the triangulation defines a piecewise-linear approximation to the image function. This approximation can be rapidly displayed using hardware Gouraud shading. The approximation is be easily updated by incrementally inserting new image samples into the triangulation [4, 9].

However, plain Delaunay triangulation does not conform to discontinuity edges in the image; as a result, many triangles are crossed by image discontinuities. Since linear interpolation is incapable of accurately approximating a discontinuous function, the corresponding sharp features appear blurred in the image. Adaptive refinement is able to reduce these artifacts at the cost of concentrating many samples in the vicinity of image discontinuities, but cannot eliminate them completely. Our approach, instead, avoids many of these artifacts while using a much smaller number of image samples. We directly compute various edges in the scene that correspond to discontinuity edges in the image plane, and represent these discontinuities explicitly as constraint edges in the triangulation. Clearly, the time spent computing discontinuity edges should be justified by their visual impact. Visible geometric edges, visible shadow edges, and visible reflections of geometric and shadow edges were chosen in our implementation: these edges can be computed with relative ease and correspond to the major discontinuities in a broad class of images.

Figure 1 illustrates our approach. Image (a) shows a simple ray-traced scene featuring a chair and its reflection in a mirror. Image (b) shows a preview reconstructed using a plain triangulation of a sparse regular sample grid (shown in (c)). All of the sharp features in the target image appear blurred in this approximation. Image (d), on the other hand, is reconstructed using the constrained triangulation shown in (e). Image (f) shows the discontinuity edges that were used in the constrained triangulation. Although both images were reconstructed using roughly the same number of samples, image (d) is clearly superior to (b).

Our algorithm starts by producing an initial preview, based on a sparse regular grid of samples and the pre-computed discontinuity edges. At this point, we must decide where additional samples should be placed in order to refine the approximation further. To solve

This paper is organized as follows. In the remainder of this section we give a brief survey of relevant previous work. In section 2 we provide a detailed description of our algorithm. In section 3, we describe the computation of various discontinuity edges in the image. In section 4, we present experiments illustrating the effectiveness of our approach.

1.1 Related Work

The particular problem of generating progressive previews has not received as much attention in the ray-tracing literature as other aspects of ray tracing. However, several techniques that have been developed to accelerate ray tracing and to allow certain interactive manipulations of ray-traced scenes are applicable to generation of progressive previews.

Breadth-first ray tracing: Hanrahan [5] suggests ray-tracing scenes in breadth-first order (instead of the usual depth-first order) to better exploit the coherence between rays at the same level. This computation order allows displaying the scene without any reflections or refractions at first, and adding these effects progressively as increasingly deeper levels of the ray-trees are computed.

Parameterized ray tracing: Séquin and Smyrl [13] introduce a technique for quickly recomputing ray-traced images in response to changes in light source intensities or material properties. When a scene is ray traced for the first time, each pixel is associated with an expression parameterized by all the light intensities and the surface properties. Following a modification to one of these parameters, the image can be recomputed quickly by re-evaluating the expression for every pixel. Note that this technique requires that the geometry of the scene and the locations of the light sources remain constant. Brière and Poulin [1] remove this restriction by detecting the exact portions of the image that must be recomputed after a change in the scene. Their method stores a ray tree with each pixel in addition to the parameterized expression that yields the pixel's color. However, the data-structures are view-dependent, and must be recomputed when the camera location changes.

Adaptive progressive refinement: Some free ray-tracing-based rendering systems, such as RADIANCE [15] and POV-Ray, offer simple previewing capabilities. These systems display a preview of the ray-traced image composed of blocks of constant color. These blocks are initially large, and they are progressively refined in an adaptive fashion as more rays are traced into the scene. Although this approach very rapidly yields a rough approximation to the image, a large number of rays must be traced before the finer details of the picture become discernible.

Interpolation: Painter and Sloan [10] propose a more sophisticated variant of adaptive progressive refinement. Their goal is to efficiently produce high quality anti-aliased images, while making the images available in a usable form early on. Their method stochastically and adaptively samples the image plane, reconstructs the image function by interpolating the samples, and filters and resamples the interpolated function for display. To interpolate the samples, Painter and Sloan suggest using Delaunay triangulation of the image plane. Our method can be viewed as an extension to their work, where discontinuity edges in the image are explicitly represented as constrained edges in the Delaunay triangulation of the image plane. This representation allows visually accurate approximations to the image function to be constructed significantly faster than would be possible with adaptive refinement alone.

Progressive Previewing of Ray-Traced Images Using Image-Plane Discontinuity Meshing

Frédéric P. Pighin[1] Dani Lischinski[2] David Salesin[1]

[1]University of Washington
Seattle, Washington 98195

[2]The Hebrew University
Jerusalem 91904, Israel

Abstract: This paper presents a new method for progressively previewing a ray-traced image while it is being computed. Our method constructs and incrementally updates a constrained Delaunay triangulation of the image plane. The points in the triangulation correspond to all of the image samples that have been computed by the ray tracer, and the constraint edges correspond to various important discontinuity edges in the image. The triangulation is displayed using hardware Gouraud shading, yielding a piecewise-linear approximation to the final image. Texture mapped surfaces, as well as other regions in the image that are not well approximated by linear interpolation, are handled with the aid of hardware texture mapping.

1 Introduction

Designing a high-fidelity photorealistic image of a 3D scene typically requires many iterations. In each iteration the designer might adjust the viewing parameters, change the positions and the intensities of the light sources, change the positions and the material properties of objects, and experiment with various renderer-dependent parameters. The scene is then re-rendered to evaluate the visual effect of these modifications. This iterative trial-and-error process is quite time-consuming, particularly when a high-quality software renderer, such as a ray-tracer, is used. The goal of the work described in this paper is to speed up the design process by allowing the designer to progressively preview the resulting image as it is being rendered.

A simple previewer that quickly displays the scene using the available graphics hardware suffices for verification of viewing parameters and object positions in the scene. However, many complex illumination phenomena are still beyond today's graphics hardware capabilities (e.g., anisotropic reflection, caustics, etc.). Thus, accurate previewers for verification of various shading parameters cannot rely on hardware rendering alone.

In this paper, we present a new previewing technique for ray-traced images. Our method utilizes hardware rendering along with software ray tracing to generate a progressive preview of the final image based on the progressively increasing number of ray-traced samples on the image plane. A Delaunay triangulation of the samples is used to construct a piecewise-linear interpolant to the image, which is displayed using hardware Gouraud shading. The key idea in our approach is to improve the accuracy of the interpolant by constraining the triangulation to contain certain discontinuity edges present in the image. Texture mapped surfaces, as well as other regions in the image that exhibit high frequencies are not well approximated by linear interpolation, and are handled instead by utilizing hardware texture mapping.