An Algorithm for Progressive Raytracing

Okan Arıkan[1] and Uğur Güdükbay[2]

[1] Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
387 Soda Hall 1776
Berkeley, CA 94720-1776, USA
[2] Department of Computer Engineering
Bilkent University
Bilkent, 06533 Ankara, Turkey

okan@cs.berkeley.edu, gudukbay@cs.bilkent.edu.tr

Abstract. Progressive generation of images is one of the important research areas of computer graphics. Especially, when the image generation takes too much time the users want to see the progress in the rendering process. The user may decide either to continue the rendering process or stop the rendering according to the current view of the image. This may be due to the fact that either the image is produced to enough detail for the user or the user does not want to continue the rendering process for some reason. This is especially important for progressive transmission of images over the Internet. The images may be progressively transmitted and when the detail level of the image is enough for the user, the transmission process may stop. In this paper, we survey the progressive image generation techniques and present an algorithm for progressive generation of raytraced images. The algorithm utilizes a refinement technique that is similar to the one used in generating interlaced images in a progressive manner.

Keywords: progressive image generation, progressive transmission, raytracing, interlacing.

1 Introduction

Raytracing is a popular image synthesis technique which is used to generate high quality images incorporating shadows, reflections, refractions, texture mapping, etc. [5,8,20]. To address deficiencies of the basic raytracing method (viz. computational complexity and realism problems), some improvements on the basic raytracing algorithm are proposed. Some of these improvements are incorporating shadows, distributed raytracing [3], adaptive depth control [7], spatial coherence [4], first hit speed-up [19], and coupling ray tracing with other global illumination models like radiosity [17]. However, the implementation of a general raytracing algorithm is quite difficult for two reasons. The ray/surface intersection calculations are detailed and the execution time is long. Execution time can

T. Yakhno (Ed.): ADVIS 2000, LNCS 1909, pp. 248–256, 2000.

be reduced for test runs by generating images at a lower resolution than the one finally required, but this would mask errors whose effect is slight and only visible in the final resolution image.

This paper gives a short survey of the methods for generating images in a progressive manner and presents a simple yet effective algorithm for this purpose. Since raytracing calculates the color of pixels independently, the screen can be rendered in any order without compromising efficiency. Thus, the image can be calculated at an initial resolution and then can be detailed further by increasing the resolution. Since the number of rays shot is directly proportional to the total calculation time, user can see the image gaining detail in time and can know the correctness of rendering. This method gives user a chance to see the rendering in progress and gain insight into the quality of final image before waiting the rendering to stop, which may take quite long time.

2 Previous Work in Progressive Image Generation

Generating images in a progressive manner is a research area for computationally intensive rendering methods, such as ray tracing, radiosity, and direct volume visualization methods. Besides, it becomes very important for the Internet since progressive transmission of images over the network saves the users' time a lot and increases network bandwidth utilization. The users may stop the transmission process when the image is transmitted to enough detail for them. Related with this, it is also important for digital content creation for the areas of long-distance education and web-based learning where the progressive generation of educational material on the computers, which is mostly images, is critical due to time restrictions.

In this section we will summarize the progressive image generation techniques in different areas of computer graphics.

2.1 Radiosity

In radiosity there is a great amount of work: here progressive refinement radiosity is the de facto standard radiosity algorithm for generating radiosity images in a progressive manner [1]. There are also some ray tracing based radiosity solutions that utilize progressive refinement [16,17]

2.2 Volume Visualization

There is a great amount of research done especially in the field of volume visualization by ray tracing to generate direct volume visualizations in a progressive manner. Levoy reformulates the front-to-back image-order volume rendering algorithm to use adaptive termination of ray tracing [10]. Another example of rendering volumetric data using progressive refinement is by Laur and Hanrahan [9].

2.3 Raytracing

There are many algorithms for generating ray traced images in a progressive manner. Examples of these are [6,11,12,13,14]. Besides, some public-domain ray-tracing-based renderers, like RADIANCE [18], offer simple previewing capabilities that allow the generation of a rough approximation to the image very rapidly. The images are refined by tracing more rays into the scene.

Painter and Sloan [12] proposes a technique to incorporate progressive refinement and anti-aliasing to an ordinary raytracer. The algorithm aims to create a high-quality anti-aliased image quickly, whose quality is improved in a progressive manner after the initial image is shown. Their algorithm achieves progressive refinement using statistics and some data structures that allow rapid detection of badly approximated areas that need to be refined. So it is an adaptive refinement algorithm where detailed parts of the image are selected first for refinement. Their method generates the samples of the image stochastically and adaptively. During this process, the samples are evaluated to determine whether further refinement is necessary or not. The evaluation criteria eliminates the plain areas from further consideration. Then, the image is reconstructed by interpolating the samples and the image is filtered and resampled for display.

Raidl and Barth [14] modifies a raytracer for fast previewing during scene composition. Their algorithm controls a strong undersampling so that a very rough approximation can be shown in a very short time. The algorithm exploits similarities to the preceding image to render small changes im the scene. The algorithm is based on a recursive image subdivision technique that uses a heuristic priority calculation to detect changed regions of the image and to prefer them in the adaptive refinement. In this way, changed regions in the image are treated early and other parts are copied from the preceding image until the priority schedule opens them for more accurate calculation.

Haines [6] proposes a technique that first renders everything at a very low resolution and then toss some Gouraud shaded polygons on the screen. Then the algorithm looks for large differences and resolves them by starting from the middle of the picture since generally the central part of the image is the interesting part. In doing this, his algorithm checks different criteria for refinement process, like object corners, colors, shadows, etc.

Pighin et al. [13] proposes a technique for progressive previewing of raytraced images while they are computed using discontinuity meshing. Their algorithm constructs and incrementally updates a constrained Delaunay triangulation of the image plane, which is suggested in [12]. The points in the triangulation correspond to all 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, producing a piecewise-linear approximation to the final image. They handle texture mapped surfaces and other regions in the image that are not well approximated by linear interpolation with the aid of hardware texture mapping.

3 The Proposed Algorithm

In this section we propose an algorithm for progressive generation of raytraced images. The algorithm takes samples for low resolution and progressively increase the detail level (and the number of rays shot). For this purpose, we used a very similar refinement that is used in the interlaced GIF images [2]. Thus, the image is calculated line by line and leaving a fixed amount of space in between. These gaps are filled with the image of next rendered line. Then, the non-rendered gaps between lines are halved and newly rendered lines are inserted at each pass. The algorithm is not an adaptive algorithm. In other words, the image is refined uniformly at all parts. The proposed progressive raytracing algorithm is summarized in Fig. 1. Figure 2 shows a simple illustration of the progressive

```
rendered[1..number_of_scanlines] stores whether the
scanline is "raytraced" (1) or "replicated" (0)

initialize rendered array to all 0's

while (there are replicated lines) do
{
      find the largest replicating span

      raytrace the pixels on the scanline at the middle of that span

      modify the rendered array element for that scanline to "raytraced"

      replicate the raytraced pixels to the empty spans below

      modify the pointers so that the largest replicating span found at
      the beginning of this iteration is replaced by two new formed spans
}
```

Fig. 1. Progressive raytracing algortihm

rendering process. The dark pixels represent the actually calculated pixels. The remaining light gray pixels are replicated from the dark ones in order to fill the gaps. In the next pass, one of the replicated lines is chosen and rendered. Then the gaps below it is filled this new line. This process continues until no replicated line is left (all pixels in the image are calculated).

The initial number of replicated lines between actually calculated lines determine how good the initial approximation will be. Moreover, allowing horizontal gapping in addition to vertical can improve the technique.

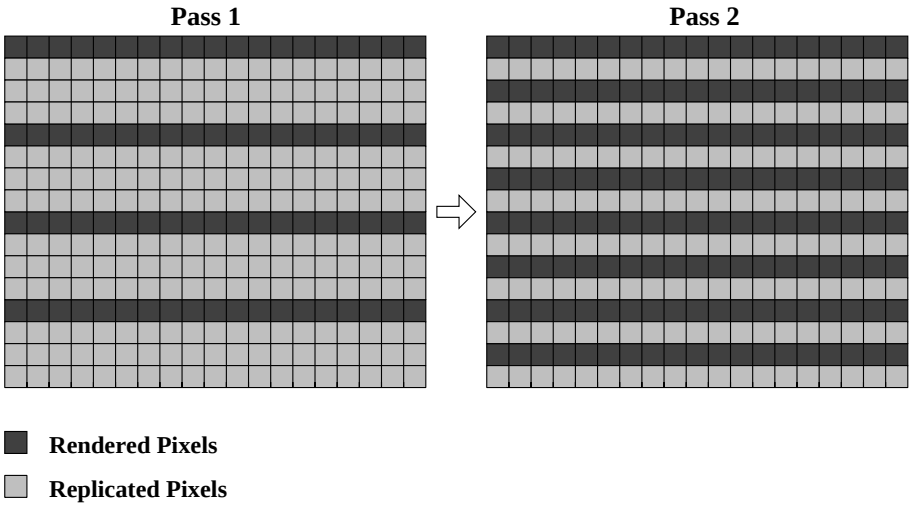

Fig. 2. Interlacing process

4 Implementation

4.1 Ray Tracer Implementation

We have based our progressive raytracer on a simple standard raytracer. This raytracer incorporates basic raytracer functionality in a modular way. Supported basic features include

- basic primitives like spheres, discs and polygons,
- reflections,
- refractions and
- texture mapping.

Although the basic raytracer also includes antialiasing routines, we removed them since antialiasing and filtering modifies pixel (or sample) color by a function of neighboring pixels (or samples). This is because progressive raytracing needs each pixel value to be calculated independently so that the pixel can be calculated in any order.

4.2 Graphical User Interface

Development of a Graphical User Interface (GUI), which will allow simultaneous management of display window and the raytracer, was the hardest part. In most of the classical window systems like Xwindows (with Toolkit), the interaction with the user is strictly event-based [15]. Since progressive raytracing requires a

raytracer core running and calculating pixels while a GUI managing the display window, we created separate processes for each part. The overall structure of the processes can be visualized like in Fig. 3. These two processes are concurrent processes which could be assigned to parallel processors.

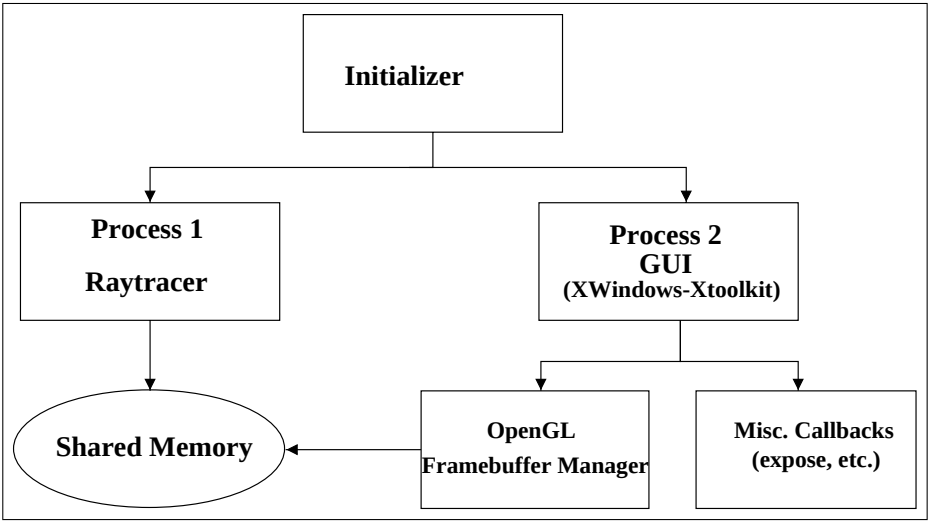

Fig. 3. Overall structure of the processes

The `Initializer` allocates necessary memory, prepares the execution environments and creates two new processes one of which is the actual raytracer and the other will be the GUI. Upon initialization, the raytracer and the GUI processes are allocated a shared memory which will hold the data. `Raytracer`, then reads the scene description and other peripheral data (like texture maps) and starts rendering and pushing data into the shared memory. The `GUI`, on the other hand, initializes the Xtoolkit and constructs the window to display the data calculated. Then, this window is associated with OpenGL[1] and callbacks are registered. One of these callbacks is a timeout mechanism which continuously generates expose callbacks in fixed intervals (2 seconds by default). Consequently, expose callback notifies OpenGL to refresh the screen from the shared memory.

5 Results

A series of raytraced images generated by our progressive raytracer is given in Fig. 4. As seen in the figure, there is no significant improvement in the image quality after 50 percent.

[1] OpenGL is a registered trademark of Silicon Graphics International, Inc.

(a) 5 % (b) 10 % (c) 25 %

(d) 50 % (e) 75 % (f) 100 %

Fig. 4. Progressive generation of a raytraced image

6 Conclusions and Future Work

We implemented a progressive version of raytracing that allows us to generate raytraced images in a progressive manner. This is useful since raytracing is a time consuming process and the images are not seen until the intensities of all pixels are generated. Sometimes it is necessary to see the low resolution versions of the final image to get an idea about the final image and discontinue the image generation process at the early stages if the final image will be useless. This is especially useful for progressive transmission of images over the Internet. Future work may include:

1. **Antialiasing:** As mentioned above, antialiasing techniques like oversampling or filtering are hard to incorporate into a progressive raytracer. Because in these methods, several samples affect the final pixel color and neighboring pixel can contribute into the final color. The final image can be computed in panels to implement antialiasing.

2. **Adaptive Refinement:** The computation power can be concentrated in parts displaying rapid changes in color (detail). This way faster convergence to the realism can be obtained.
3. **User Dictated Refinement:** A different user interface that will allow user to determine the parts to compute first can be developed, increasing the usability of the program.

Acknowledgments. This research is partially supported by an equipment grant from Turkish Scientific and Technical Research Council (TÜBİTAK) with grant number EEEAG 198E018. Thanks to Varol Akman for valuable discussions.

References

1. Cohen M.F. Chen, S.E., Wallace, J.R. and Greenberg, D.P., "A Progressive Refinement Approach to Fast Radiosity Image Generation", *ACM SIGGRAPH Conference Proceedings*, pp. 75-84, 1988.
2. CompuServe Inc., "GIF: Graphics Interchange Format", `http://www.daubnet.com/formats/GIF.html`.
3. Cook, R.L., Porter, T., and Carpenter, L., "Distributed Ray Tracing", *ACM SIGGRAPH Conference Proceedings*, pp. 137-144, 1984.
4. Glassner, A., "Space Subdivision for for Fast Ray Tracing", *IEEE Computer Graphics and Applications*, Vol. 4 No. 4, October 1984.
5. Glassner, A., *An Introduction to Ray Tracing*, Academic Press, New York, 1989.
6. Haines, E., "Progressive Ray Tracing and Fast Previews", Ray Tracing News, Vol. 10, No. 1, edited by Eric Haines, January 1997, `http://www.acm.org/tog/resources/RTNews/html`.
7. Hall, R.A., and Greenberg, D.P., "A Testbed for Realistic Image Synthesis", *IEEE Computer Graphics and Applications*, Vol. 3, No. 8, November 1983.
8. Kay, D.S., "Transparency, Refraction and Raytracing for Computer Synthesized Images", Master's Thesis, Cornell University, Jan. 1979.
9. Laur, D., and Hanranan, P., "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering", *ACM SIGGRAPH Conference Proceedings*, pp. 285-288, 1991.
10. Levoy, M., "Volume Rendering by Adaptive Refinement", *The Visual Computer*, Vol. 6, No. 1, pp. 2-7, February 1990.
11. Maillot J-L. and Carraro, L. and Peroche, B., "Progressive Ray Tracing", *Proc. of Third Eurographics Workshop on Rendering*, pp. 9-20, Bristol, UK, May 1992.
12. Painter, J. and Sloan, K., "Antialiased Ray Tracing by Adaptive Progressive Refinement," *ACM SIGGRAPH Conference Proceedings*, pp. 281-288, 1989.
13. Pighin, F., Lischinski, D., and Salesin, D., "Progressive Previewing of Ray-Traced Images Using Image-Plane Discontinuity Meshing", *Proc. of 8th Eurographics Workshop on Rendering*, pp. 115-125, France, July 1997.
14. Raidl, G. and Barth, W., "Fast Adaptive Previewing by Ray Tracing", *Proc. of 12th Spring Conference on Computer Graphics*, edited by W. Purgathofer, pp. 247-255, Comenius University, Bratislava, Slovakia, June 1996.
15. Scheifler, R.W., Gettys, J. and Newman, R. *X Window System*, Digital Press, 1988.

16. Schlick, C. and Le Saëc, B., "A Progressive Ray Tracing based Radiosity with General Reflectance Functions", *Proc. of Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pp. 101-113, June 1990.
17. Wallace, J.R., Elmquist, E.A., Haines, E.A., "A Ray Tracing Algorithm for Progressive Radiosity", *ACM SIGGRAPH Conference Proceedings*, pp. 315-324, 1989.
18. Ward, G., "The RADIANCE Lighting Simulation and Rendering System", *ACM SIGGRAPH Conference Proceedings*, pp. 459-472, 1994.
19. Weghorst, H., Hooper, G, and Greenberg, D.P., "Improved Computational Methods for Ray Tracing," *ACM Transactions on Graphics*, Vol. 3, No. 1, pp. 52-69, 1984.
20. Whitted, T., An Improved Illumination Model for Shaded Display", *Communications of ACM*, Vol. 23, No. 6, pp. 343-349, June 1980.