Temporally Coherent Interactive Ray Tracing

W. Martin S. Parker E. Reinhard
P. Shirley W. Thompson

UUCS-01-005

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

May 15, 2001

Abstract

Although ray tracing has been successfully applied to interactively render large datasets, supersampling pixels will not be practical in interactive applications for some time. Because large datasets tend to have subpixel detail, one-sample-per-pixel ray tracing can produce visually distracting popping and scintillation. We present an algorithm that directs primary rays toward locations rendered in previous frames, thereby increasing temporal coherence. Our method tracks intersection points over time, and these tracked points are used as an oracle to aim rays for the next frame. This is in contrast to traditional image-based rendering techniques which advocate color reuse. We so acquire coherence between frames which reduces temporal artifacts without introducing significant processing overhead or causing unnecessary blur.

Temporally Coherent Interactive Ray Tracing

W. Martin S. Parker E. Reinhard P. Shirley W. Thompson
University of Utah www.cs.utah.edu

Abstract. Although ray tracing has been successfully applied to interactively render large datasets, supersampling pixels will not be practical in interactive applications for some time. Because large datasets tend to have subpixel detail, one-sample-per-pixel ray tracing can produce visually distracting popping and scintillation. We present an algorithm that directs primary rays toward locations rendered in previous frames, thereby increasing temporal coherence. Our method tracks intersection points over time, and these tracked points are used as an oracle to aim rays for the next frame. This is in contrast to traditional image-based rendering techniques which advocate color reuse. We so acquire coherence between frames which reduces temporal artifacts without introducing significant processing overhead or causing unnecessary blur.

1 Introduction

Improvements in general-purpose computer systems have recently allowed ray tracing to be used in interactive applications [10, 13, 18]. On some applications such as isosurface rendering [14], ray tracing is superior to even highly optimized z-buffer methods. In this paper we present an algorithm to reduce dynamic artifacts in an interactive ray tracing system. This algorithm is relatively simple, and attacks what we believe is the dominant visual artifact in current interactive ray tracing systems.

Because ray tracing has a computational cost roughly linear in the number of samples (primary rays), interactive implementations have used one primary ray per pixel and relatively small image sizes, with 512 by 512 images being common. As raw processor performance increases, these systems can improve by creating larger images, increasing frame rate, or by taking multiple samples per pixel. Once the framerate is at interactive levels (approximately 15 frames per second or higher), our experience is that increasing the number of pixels is usually a better option than using multiple primary rays per pixel. Current systems that ray trace large datasets are approximately one hundred times too slow to render to the highest resolution screens at thirty frames per second[1]. This suggests that even if Moore's Law continues to hold for another ten years, we will still be taking only one primary ray per pixel until at least that time.

Unfortunately, one sample per pixel ray tracing introduces aliasing issues which need to be addressed. Most anti-aliasing approaches were developed in an era where the available processing power allowed only relatively simple scenes to be rendered and displayed. As a consequence, the ratio of polygons to pixels was low, usually resulting in long straight edges. Aliasing issues are particularly evident in such images, revealing themselves in the form of jaggies and Moiré patterns [1, 2]. Animations of such scenes cause the running ants effect along edges and visually distracting scintillations when texture mapping is applied.

[1] Recent advances in display technology have produced LCD displays with approximately 9 million pixels, which is a factor of 3 to 8 higher than currently common.

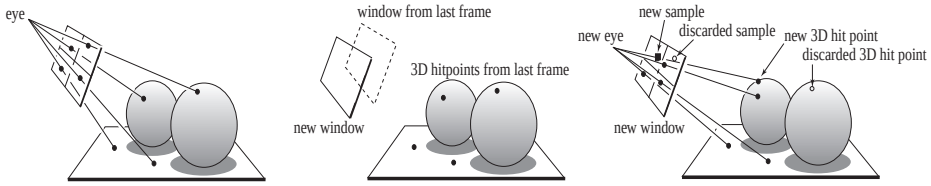

Fig. 1. *Point tracking algorithm.*

Two solutions are common: spatially filtering the scene before sampling, or spatially filtering the image after sampling [5, 9, 11, 12]. Pre-filtering the scene, e.g. using LOD management or MIP-mapping, may not always be possible, as is the case for the 35 million sphere crack propagation dataset used in Parker et al. [14]. Super-sampling the image is considered too costly for interactive systems, as argued above. Another trade-off exists in the form of stochastic sampling to turn aliasing into visually less offensive noise [4, 6]. A by-product of spatial filtering is that temporal artifacts are also reduced, although temporal anti-aliasing methods have received some attention [8, 16].

Since the development of these anti-aliasing schemes, two things have been changing. Scene complexity now commonly far exceeds pixel resolutions and display devices are about to become available at much higher resolutions. The combination of these two leads to the presence of an approximate continuum of high frequencies in the image, which causes spatial aliasing to be visually masked [3, 7]. Whenever this occurs, spatial filtering produces images that contain more blur than necessary. Higher scene complexity and device resolution do not tend to mask temporal aliasing artifacts as well. A good solution to these issues would therefore preserve spatial crispness while at the same time minimizing temporal artifacts.

Our approach is to revert to point sampling where sample locations in the current frame are guided by results from previous frames. Such guidance introduces temporal coherence that will reduce scintillation and popping. It is also computationally inexpensive so that most of the processing power can be devoted to producing point samples of highly complex scenes. Scene complexity in turn masks spatial aliasing. The method is demonstrated using an interactive ray tracing engine for complex terrain models.

2 Point tracking algorithm

For high polygon to pixel ratios, the scene is sparsely sampled. In interactively ray traced animations, a brute-force approach is likely to possess little temporal coherence. With high probability, each pixel will have different scene features projected onto them from frame to frame. This causes popping/scintillation which is visually distracting.

This effect can be minimized by using an algorithm which projects the same feature within a region of a scene onto the screen over the course of a number of frames. We therefore seek a method which keeps track of which points in the scene were sampled during previous frames. This can be easily accomplished within a ray tracing framework by storing the intersection points of the primary rays with their corresponding pixels. Prior to rendering the next frame, these intersection points are associated with new pixels using the perspective projection determined by the modified camera parameters. Instead of painting the reprojected points directly on the screen [19], which may lead to inaccuracies, this reprojection serves the sole purpose of directing new rays towards objects that have been previously hit.

After the reprojection of the last frame's intersection points, for each pixel one of three situations may arise:

- One intersection point was projected onto a pixel. This is the ideal case where scintillation is avoided.
- Two or more intersection points were projected onto a single pixel. Here, an oracle is required to choose which of the points is going to be used. We use a simple heuristic which chooses the point with the smallest distance to the camera origin. Points further away are more likely to be occluded by intervening edges and are therefore less suitable candidates.
- Zero points were projected onto a pixel. For such pixels no guidance can be provided and a regular or jittered ray is chosen instead.

For each pixel a new ray is traced. Instead of using image space heuristics to choose ray directions, an object space criterion is used: we aim for the intersection points that were projected onto each pixel. A new regular or jittered ray direction is chosen only in the absence of a reprojected point for a given pixel.

Once stored intersection points have been used for the current frame, they are discarded. The ray tracing process produces shaded pixel information for the current frame as well as a list of intersection points to be used for the next frame.

3 Results

To demonstrate the effectiveness of this point tracking method, we use the interactive ray tracer of Parker et. al. [13] to render two different terrain models, which are called Hogum and Range-400. Images of these data sets are given in Figure 2 and their geometric complexity is given in Table 1. The intersection routine is similar to that used for isosurface rendering [14]. The height field is stored in a multiply nested 2D regular grid with the minimum and maximum heights recorded for each grid cell. When a leaf cell is an intersection candidate, the exact intersection is computed with the bilinear patch defined by the four points. The measurements presented in this paper are all collected using a 32 processor SGI Origin 3800. Each processor is an R12k running at 400 MHz. We used 30 processors for each run plus an additional processor running the display thread.

The computational cost of the point tracking system is assessed in Figure 3. In this graph, we compare the interactive ray tracer with and without point tracking for different image resolutions. For both data sets it appears that satisfactory frame-rates can be achieved for image sizes up to 512^2 pixels.

The performance penalty of using the point tracking algorithm is roughly 20%, which is small enough to warrant its use. Although we compare with point sampled images, to obtain similar (but more blurred) results, one would have to compare its performance with super-sampled images, whose cost is linear in the number of samples taken per pixel. Both single sample images with and without point tracking are vastly less expensive than super-sampled images.

The visual appearance of the animations with and without point tracking are presented in the accompanying video[2]. Assuming a worst-case scenario, the point tracking algorithm is most effective if a single intersection point projects onto a given pixel. This

[2]The video is in SVHS format and is captured using the direct SVHS output of the Origin system. This does cause some blurring and also some extra popping. However, the relative magnitude of the visual artifacts is approximately what we observe on the actual display.

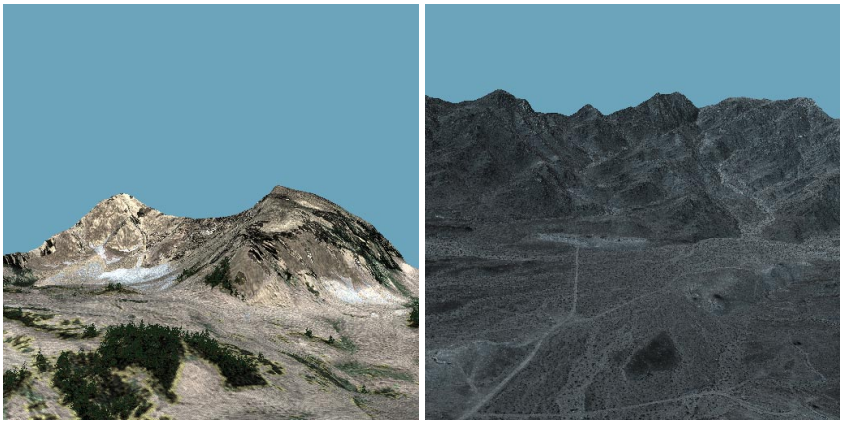

Fig. 2. *Hogum and Range-400 terrain datasets.*

Terrain	Resolution	Texture resolution
Hogum	175×199	1740×1980
Range-400	3275×2163	3275×2163

Table 1. *Terrain complexity*

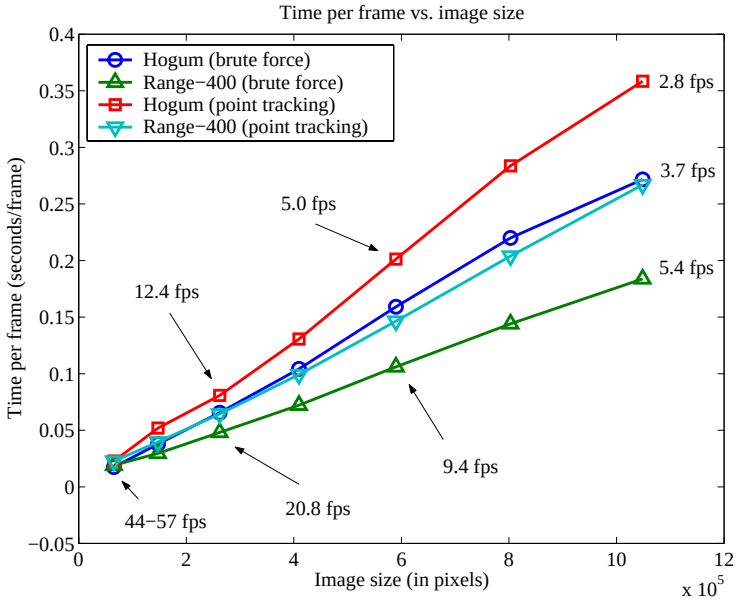

Fig. 3. *Hogum and Range-400 data sets rendered with and without point tracking at image resolutions of* 256^2, 384^2, 512^2, 640^2, 768^2, 896^2, *and* 1024^2 *pixels. Shown is the average frame-rate achieved for each animation.*

	Intersection points per pixel				
Terrain	0	1	2	3	4
Hogum	21.93%	58.22%	18.18%	1.63%	0.04%
Range-400	19.68%	64.73%	14.21%	1.33%	0.05%

Table 2. *Percentage of pixels that have 0 to 4 intersection points reproject to them. The ideal case is where 1 intersection point projects to a pixel.*

is when perfect frame-to-frame coherence is achieved. When zero points project to a pixel, a ray cast for that pixel is likely to produce a contrast change, causing scintillation. When more than one point reprojects to a single pixel, some information will locally disappear for the next frame. This may also lead to scintillation.

For the above experiments, roughly 60% of the pixels in each frame are covered by exactly one intersection point (Table 2). In the remaining cases, either zero or more than one intersection point reprojects to the same pixel. Hence, one could argue that our algorithm presents a 60% visual improvement over standard point sampled animated imagery. Note that these numbers are dependent on both the model and camera speed.

4 Conclusions

Our paper has assumed that multisampled spatial filtering will not soon be practical for interactive ray tracing. Anticipating a trend towards higher resolution display devices, it is also likely that spatial filtering to achieve anti-aliasing will become less necessary, because the effect of spatial aliasing will largely be masked. However, the same is not true for temporal aliasing. Combining the performance advantage of point sampling over super-sampling with a mechanism to reduce temporal aliasing, we have achieved visually superior animations at a cost that is only slightly higher than point sampling alone. Moreover, its computational expense is much less than the cost of conventional anti-aliasing schemes such as super-sampling. Using this point tracking algorithm, the increase in computational power which is to be expected according to Moore's Law can be spent on rendering larger images, rather than on expensive filtering operations.

We speculate that, in the future, the point tracking technique may be applied independent of the reconstruction used for display. If we allow tracked points to persist even when they co-occupy a pixel, then more sophisticated reconstruction methods could be applied to increase point reuse. This would be in the spirit of Simmons et al. [17] and Popescu et al. [15]. We have not pursued this in the present work, noting that the former use sparser sample sets than we do, and the latter technique requires special purpose hardware. Nonetheless, we believe progress in computational performance will continue to outpace improvements in screen resolution, and conclude that such sophisticated reconstruction using tracked points will eventually be possible.

Acknowledgments

The authors would like to thank Brian Smits for insightful discussions. This work was supported in part by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219), and NSF grants 97-31859, 99-78099, and 97-20192. All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

1. BLINN, J. F. Return of the jaggy. *IEEE Computer Graphics and Applications 9*, 2 (March 1989), 82–89.
2. BLINN, J. F. What we need around here is more aliasing. *IEEE Computer Graphics and Applications 9*, 1 (January 1989), 75–79.
3. BOLIN, M. R., AND MEYER, G. W. A perceptually based adaptive sampling algorithm. *Proceedings of SIGGRAPH 98* (July 1998), 299–310. ISBN 0-89791-999-8. Held in Orlando, Florida.
4. COOK, R. L. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (January 1986), 51–72.
5. CROW, F. C. A comparison of antialiasing techniques. *IEEE Computer Graphics and Applications 1*, 1 (January 1981), 40–48.
6. DIPPÉ, M. A. Z., AND WOLD, E. H. Antialiasing through stochastic sampling. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), B. A. Barsky, Ed., vol. 19, pp. 69–78.
7. FERWERDA, J. A., PATTANAIK, S. N., SHIRLEY, P., AND GREENBERG, D. P. A model of visual masking for computer graphics. *Proceedings of SIGGRAPH 97* (August 1997), 143–152. ISBN 0-89791-896-7. Held in Los Angeles, California.
8. GRANT, C. W. Integrated analytic spatial and temporal anti-aliasing for polyhedra in 4-Space. In *Computer Graphics (SIGGRAPH '85 Proceedings)* (July 1985), B. A. Barsky, Ed., vol. 19, pp. 79–84.
9. MITCHELL, D. P. Generating antialiased images at low sampling densities. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), M. C. Stone, Ed., vol. 21, pp. 65–72. held in Anaheim, California; 27 – 31 July 1987.
10. MUUSS, M. J. *towards real-time ray-tracing of combinatorial solid geometric models. In* Proceedings of BRL-CAD Symposium *(June 1995).*
11. NORTON, A., ROCKWOOD, A. P., AND SKOLMOSKI, P. T. *Clamping: A method of antialiasing textured surfaces by bandwidth limiting in object space. In* Computer Graphics (SIGGRAPH '82 Proceedings) *(July 1982), vol. 16, pp. 1–8.*
12. PAINTER, J., AND SLOAN, K. *Antialiased ray tracing by adaptive progressive refinement. In* Computer Graphics (SIGGRAPH '89 Proceedings) *(July 1989), J. Lane, Ed., vol. 23, pp. 281–288. held in Boston, Massachusetts; 31 July – 4 August 1989.*
13. PARKER, S., MARTIN, W., SLOAN, P.-P. J., SHIRLEY, P., SMITS, B., AND HANSEN, C. *Interactive ray tracing.* 1999 ACM Symposium on Interactive 3D Graphics *(April 1999), 119–126.*
14. PARKER, S., PARKER, M., LIVNAT, Y., SLOAN, P.-P., HANSEN, C., AND SHIRLEY, P. *Interactive ray tracing for volume visualization. In* IEEE Transactions on Visualization and Computer Graphics *(July-September 1999).*
15. POPESCU, V., EYLES, J., LASTRA, A., STEINHURST, J., ENGLAND, N., AND NYLAND, L. *The WarpEngine: An architecture for the post-polygonal age.* Proceedings of SIGGRAPH 2000 *(July 2000), 433–442. ISBN 1-58113-208-5.*
16. SHINYA, M. *Spatial anti-aliasing for animation sequences with spatio-temporal filtering. In* Computer Graphics (SIGGRAPH '93 Proceedings) *(1993), J. T. Kajiya, Ed., vol. 27, pp. 289–296.*
17. SIMMONS, M., AND SÉQUIN, C. H. *Tapestry: A dynamic mesh-based display representation for interactive rendering.* Rendering Techniques 2000: 11th Eurographics Workshop on Rendering *(June 2000), 329–340. ISBN 3-211-83535-0.*
18. WALD, I., SLUSALLEK, P., BENTHIN, C., AND WAGNER, M. *Interactive rendering with coherent ray tracing. In* Eurographics 2001 *(2001).*
19. WALTER, B., DRETTAKIS, G., AND PARKER, S. *Interactive rendering using the render cache. In* Rendering Techniques '99 *(1999), D. Lischinski and G. W. Larson, Eds., Eurographics, Springer-Verlag Wien New York, pp. 19–30.*