Temporally Coherent Interactive Ray Tracing

William Martin† Erik Reinhard‡ Peter Shirley†
Steven Parker† William Thompson†

†School of Computing, University of Utah
‡School of Electrical Engineering and Computer Science, University of Central Florida

Abstract

Ray tracing exhibits visible temporal aliasing artifacts in interactive viewing of complex datasets. We present a technique to reduce scintillation in point sampled imagery by targeting new rays to intersection points from previous frames. Remaining artifacts are abated by blending between successive frames. The algorithm achieves a reduction in scintillation comparable to that of supersampling, but at a significantly lower cost. Sample animations are available online.

1 Introduction

Improvements in general-purpose computer systems have recently allowed ray tracing to be used in interactive applications [2, 3, 6–8]. For some applications such as isosurface visualization [3] or rendering complex static geometry [8], ray tracing is superior to even highly optimized conventional z-buffer methods. Nonetheless, current processors are too slow to interactively ray trace the most detailed data sets to the highest resolution displays [2, 3, 8]. To achieve interactivity, current systems use a sparse sampling of the image (approximately one ray per pixel) and relatively small image sizes, with 512 by 512 resolutions being common. There is a continuing trend towards higher resolution displays and more complex models, so high-resolution multisampled interactive ray tracing will not be feasible in the near future. Current systems would need more than a one-thousand times increase in number of rays per frame to achieve HDTV resolution with moderate antialiasing. In this paper we assume that we are limited to only a few samples per pixel, and that improvements in processing power will be targeted towards higher resolution rather than better antialiasing. Our application is ray tracing scenes of high geometric or textural complexity at interactive rates.

A problem with large models is that they tend to cause sub-pixel detail in rendered images, leading to temporal aliasing. We believe this is the dominant artifact in current interactive ray tracing systems. If we are restricted to a few rays per pixel, spatially filtering the scene before sampling is a common strategy. However, 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 visualized via ray tracing (Figure 1).

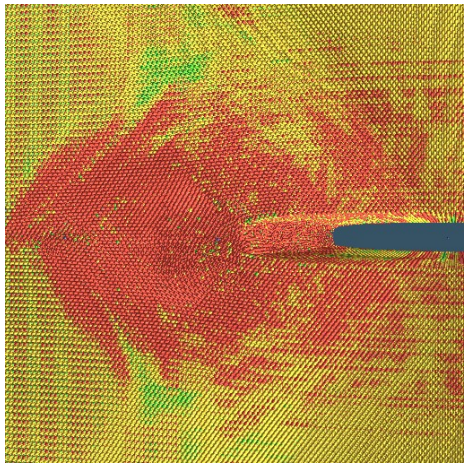

Figure 1: A 35 million sphere scientific dataset.

Our approach to reduce temporal artifacts 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, costing significantly less than adding one extra sample to each pixel. The method is demonstrated using an interactive ray tracing engine for complex terrain and visualization datasets.

2 Point tracking algorithm

The temporal aliasing in interactively ray traced sequences arises because each pixel will likely correspond to different scene features from one frame to the next. This effect can be minimized by using an algorithm which tracks a consistent set of features over the course of a number of frames. We therefore seek a method which targets scene points sampled during previous frames. This can be accomplished by storing the intersection points of the primary rays with their corresponding pixels. Prior to rendering the next frame, the camera parameters are updated, and the hit points are mapped to new pixels via backprojection onto the image plane. This projection has been previously exploited by IBR researchers [1]. However, instead of painting the reprojected points directly on the screen [9], which may lead to inaccuracies, this reprojection serves the sole purpose of directing new rays towards objects that have been previously hit.

The reprojection of the previous frame's intersection points gives rise to one of three possible situations: zero, one, or many points are projected to any given pixel. If zero points are projected onto a pixel, no guidance can be provided and a new ray is created. If one intersection point is projected onto a pixel, a new ray is sent towards it. If two or more intersection points are projected onto a single pixel, then we use a simple heuristic which chooses the point with the smallest distance to the camera

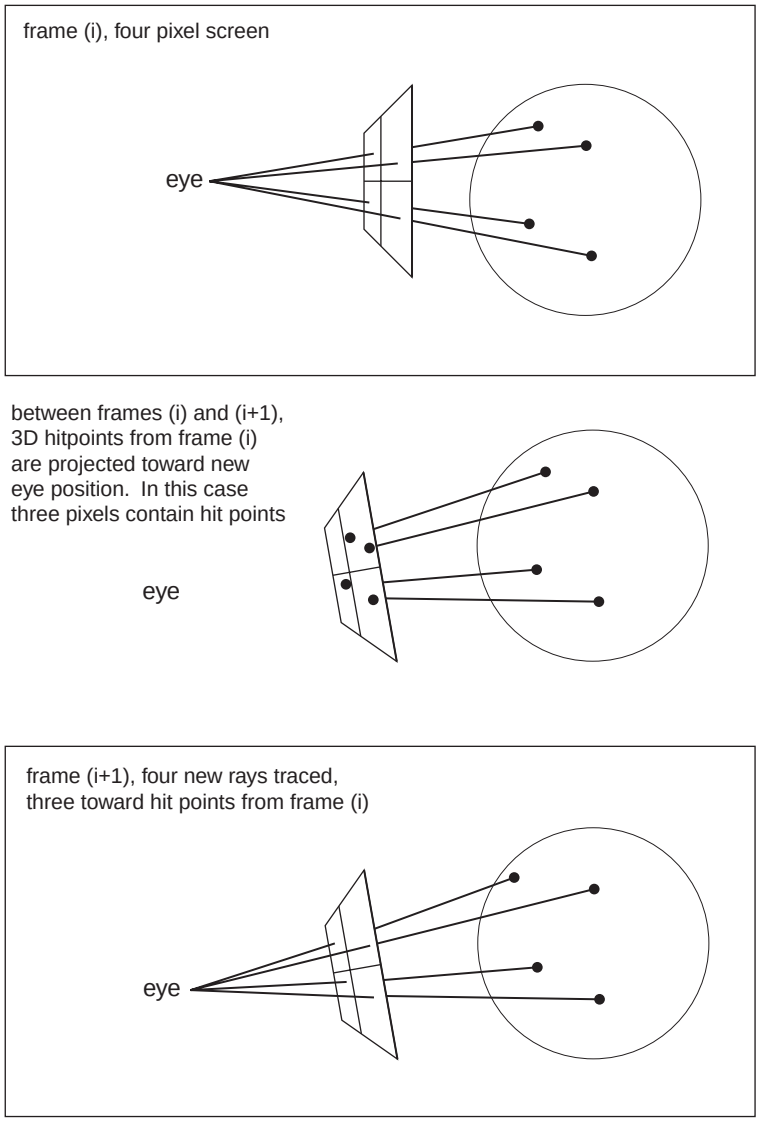

Figure 2: Overview of the point tracking algorithm: In the top panel, rays are traced through a four pixel screen, and the 3D hit positions are cached. In the middle panel, the eye position changes, and the cached points are back-projected onto the new viewing plane to associate them with screen pixels. Two of the cached points are associated with the top right pixel, whereas no points map to the top left pixel. In the bottom panel, a new ray is fired through each pixel, targeting cached points whenever possible. The closest of the two hit points is targeted for the top right pixel. Lacking target points, a new ray is randomly generated for the top left pixel.

origin. Points farther away are more likely to be occluded by intervening edges and are therefore less suitable candidates. 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. This idea is illustrated in Figure 2.

The method of tracking points across frames reduces the appearance of scintillation in animations of ray traced scenes. However, some scintillation will remain, because not all pixels will be associated with a unique tracked point as noted above. The question of whether the remaining artifacts are visually objectionable depends largely on the dataset. We have found that the visible aliasing in our scenes is typically due to high-frequencies in the temporal dimension. It can therefore be reduced using a narrow temporal filter. In our case, we have chosen a box filter which spans three frames. We extend our basic algorithm to cache the three most recent frames, each generated using the point tracking algorithm described above. The image displayed is then the per pixel average of the color values stored in these three frames. (The number three is heuristic, and may need to be adjusted depending on the framerate and dataset.) Because our blend is based on frames previously rendered, the point tracking augmented with the temporal blend is essentially no slower than the point tracking algorithm by itself. In our system, temporal blending may be toggled off by the user if satisfactory results are obtained using point tracking alone. The method of tracking points is summarized in Algorithm 1.

Algorithm 1 Point Tracking Algorithm

```
for each Frame i do
    Update the camera position
    Reproject hit points from the Frame i-1
    for each Pixel j do
        Fire ray, targeting a projected hit point when available
        Cache new hit point and color with Frame i
    end for
    Average the colors from the last three frames to generate a screen image
end for
```

3 Results

To demonstrate the effectiveness of our point tracking method, we use an interactive ray tracer to render two terrain models, which are called Hogum and Range-400 (Figure 3), and the crack propagation dataset of Figure 1. The measurements presented in this paper were all collected using a 32 processor SGI Origin 3800. Each processor was an R12k running at 400 MHz. We used 30 processors for each run plus one additional processor running the display thread. The reprojection phase was performed in parallel, with each processor operating on a contiguous region of the image plane.

The runtime of the point tracking system is between 1.25 and 1.45 times the runtime of conventional ray tracing for all datasets, image sizes, and camera positions we have tested, with 1.35 times being typical. Most of this cost is due to the reprojection at the beginning of each frame. To put this penalty in perspective, the cost of 4 times

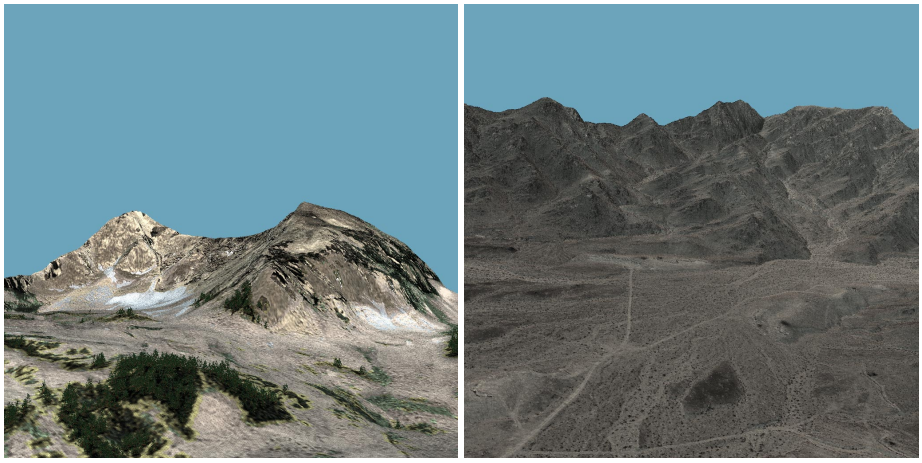

Figure 3: *Top: Hogum dataset with* 175×199 *height values and a* 1740×1980 *texture. Bottom: Range-400 dataset with* 3275×2163 *height values and a texture of the same resolution.*

supersampling is 4 times the runtime. The point tracking memory overhead is two buffers of points and three buffers of RGB colors. Each of these buffers has the same number of elements as the number of pixels in the image.

The point tracking algorithm is most effective if a single intersection point projects onto a given pixel. This 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 from the next frame. This may also lead to scintillation.

To validate our technique we have compared point tracked animations with untracked animations generated by a) point sampling, b) spatial supersampling, and c) temporal blending (each displayed pixel value is an average over three consecutive frames). The reader is referred to the web sites listed at the end of this paper for animations demonstrating the technique. Our opinion is that the subjective effectiveness of our approach in removing visible flicker is comparable to that of supersampling, but without the corresponding degree of blur. This opinion is reinforced by the data presented in Table 1. For typical interactive walkthroughs, roughly 60% of the pixels in each frame are covered by exactly one intersection point. So for a majority of the pixels in each frame, temporal coherence is successfully exploited.

4 Discussion

The technique presented is ideally suited for scenes possessing large areas of subpixel detail. In such scenes our experiments using point tracking have demonstrated a marked reduction in scintillation. We reiterate that the technique targets temporal

	Intersection points per pixel				
Dataset	0	1	2	3	4
Hogum	21.9%	58.2%	18.2%	1.6%	0.0%
Range-400	19.6%	64.7%	14.2%	1.3%	0.1%
Spheres	20.6%	60.6%	17.1%	1.6%	0.0%

Table 1: *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.*

aliasing, and is not prescribed as a solution to the problem of spatial aliasing. A valid concern is the degree to which the biased sampling induced by point tracking might introduce artifacts. While in principle this could occur, we have not experienced significant problems for the wide variety of scenes tested. The artifacts introduced by point tracking are nearly identical to those introduced by jittering. Edge fidelity suffers slightly, but no more than a pixel width. Spatial aliasing is transformed into noise.

We have 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 the temporal aliasing artifacts which this paper addresses. Our current implementation is not helpful for specular reflections, and handling specular surfaces correctly would be difficult. This has not been a problem for us because our primary applications are terrain and scientific visualization, where specular effects are rare.

We speculate that the point tracking technique may in the future be applied independently 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, in the spirit of Simmons et al. [5] and Popescu et al. [4]. We have not pursued this in the present work, noting that the former technique uses sparser sample sets than we do, and the latter requires special purpose hardware.

5 Acknowledgments

The authors would like to thank Brian Smits for insightful discussions, and Richard Coffey and Robert Cummins for their tireless technical support. This work was supported in part by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (EIA-89-20219), and NSF grants IIS-0080999, CCR-9731859, 9977218, and 9978099. 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] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings,

Annual Conference Series, pages 39–46, Los Angeles, California, August 1995. ACM SIGGRAPH / Addison Wesley. ISBN 0-201-84776-0.

[2] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter S. Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 119–126. ACM SIGGRAPH, April 1999. ISBN 1-58113-082-1.

[3] Steven Parker, Michael Parker, Yarden Livnat, Peter-Pike Sloan, Charles Hansen, and Peter Shirley. Interactive ray tracing for volume visualization. In *IEEE Transactions on Visualization and Computer Graphics*, volume 5, pages 238–250, July-September 1999.

[4] Voicu Popescu, John Eyles, Anselmo Lastra, Joshua Steinhurst, Nick England, and Lars Nyland. The WarpEngine: An architecture for the post-polygonal age. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 433–442. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, July 2000. ISBN 1-58113-208-5.

[5] Maryann Simmons and Carlo Séquin. Tapestry: A dynamic mesh-based display representation for interactive rendering. In B. Péroche and H. Rushmeier, editors, *Rendering Techniques 2000 (EG Workshop on Rendering 2000)*, pages 329–340, Brno, Czech Republic, June 2000. Eurographics, Springer Wien.

[6] Ingo Wald and Philipp Slusallek. State of the art in interactive ray tracing. In *State of the Art Reports, EUROGRAPHICS 2001*, pages 21–42. EUROGRAPHICS, Manchester, United Kingdom, September 2001.

[7] Ingo Wald, Philipp Slusallek, and Carsten Benthin. Interactive distributed ray-tracing of highly complex models. In S. J. Gortler and K. Myszkowski, editors, *Rendering Techniques 2001 (EG Workshop on Rendering 2001)*, pages 274–285, London, United Kingdom, June 2001. Eurographics, Springer Wien.

[8] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153–164, 2001.

[9] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In D. Lischinski and G. W. Larson, editors, *Rendering Techniques '99 (EG Workshop on Rendering 1999)*, pages 19–30, Granada, Spain, June 1999. Eurographics, Springer Wien.

Web Information:
Animations demonstrating the technique are available via:
`http://www.acm.org/jgt/papers/MartinEtAl2002`
or directly on the authors' web site at:
`http://www.cs.utah.edu/~wmartin/PointTracking`.