Using Quad-Trees for Acceleration of Physically-Based Image-Space Rendering of Glare

Peter Sikachev, Ilya Tisevich, Alexey Ignatenko
Department of Computational Mathematics and Cybernetics
Moscow State University, Moscow, Russia
{psikachev, itisevich, ignatenko}@graphics.cs.msu.ru

Abstract

Glare effects have recently become a point of interest for many researchers. It adds realism to a 3D-scene, making light sources and bright objects look true-to-life. Several applications demand physical proof for these effects.

Though most of the methods are only concerned with a simple "bloom" effect, new methods with more complex effects become available, taking in consideration such parameters as lens aperture and occluder. Some of them are based on wave optics, but there are also a number of methods for real-time hardware-accelerated rendering.

An interactive method for glare rendering is proposed, based on Fourier Optics. Several improvements were made in speed and physical correctness of our method. We provide results of a comparison between real-life and simulated effects showing realism of our approach.

Keywords: *Fraunhofer Diffraction, Streaks, Glare, Bloom, HDR Rendering, Hardware-Accelerated Rendering.*

1. INTRODUCTION

Rendering of high intensity light is a tricky problem because of physical limitations in the brightness of displays and other output devices. Therefore techniques like bloom and streaks appeared which could reproduce the effects of high-intensity lighting, simulating genuine optical effects on limited brightness hardware, such as computer displays. These effects include interaction of observed light with eyelashes, diaphragm, retina and camera film/matrix.

Bloom simulates the retina's feature to illuminate the neighboring cells to those which bright light has effectively hit. Streaks are used to visualize the diffraction, which appears due to the presence of an occluder, that is far away (Fresnel diffraction, e. g. streaks from glass) or near the viewpoint (Fraunhofer diffraction, e. g. streaks from eyelashes or finite lens aperture).

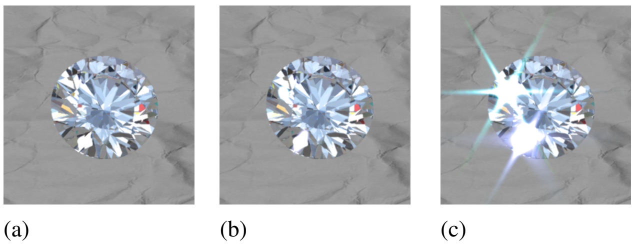

Figure 1: Example renderings with different HDR effects. (a) no HDR effect; (b) bloom effect (c) Streaks and bloom effects

In our work we limit ourselves to rendering Fraunhofer diffraction for an arbitrary scene and given lens or eye properties. Methods we used are similar to [1].

Our contribution is acceleration of the physically-based streak rendering algorithm. Acceleration rate varies from 1.5 to 4 times, depending on the selected algorithm.

2. RELATED WORK

Several algorithms have been developed to render bloom and streak effects.

Methods [7, 8] render bloom using simple two or more pass algorithm. They are hardware-accelerated and, being very fast, are used in many applications like computer games and demonstrations.

Method [3] simulates the complex effect on the retina that depends on many parameters, such as scene illumination and age of the observer. It doesn't take into consideration any effects linked with diffraction in an occluder or a lens.

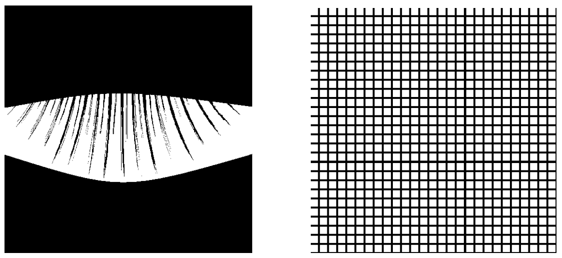

Figure 2: Occluder samples for an eye (left) and camera (right, objective polishing is simulated by regular grid) cases. Eye occluder simulates eyelashes, camera occluder simulates rough surface of a "streak" optical filter.

Methods such as [2, 5] are suited for hardware-accelerated fast visualization. They somehow lack physical proof, but work fine for applications, where 'true' photorealism is not required.

Finally, there are methods like [1] which are specifically suited for rendering Fraunhofer diffraction streaks. Authors describe how this method could be used in real-time, but in most cases it is too time-consuming even for high-end GPUs.

Section 3 describes how we integrate this method into our rendering pipeline. There are some interesting tricks about how we combine streaks and make several improvements to the algorithm (increasing its speed), which are described in Section 4. We conclude and describe our future work plans in Section 6.

3. STAR RENDERING PIPELINE

3.1 Streak Rendering Stages

In a nutshell, streak rendering technique consists of two stages (see Figure 3).

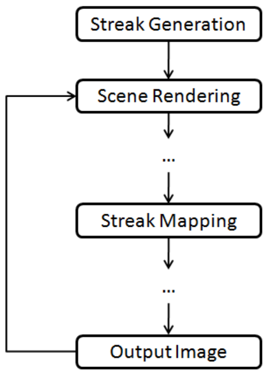

Figure 3: Streak rendering stages in the graphics pipeline.

At the first stage, a streak image is generated using occluder and lens images exactly as in [1]. This is done once, until any change in the occluder or the lens takes place; recalculation is not needed at each frame.

At the second stage we map this image on the bright points of the image, using additive blending [4]. Unlike [1], we cannot afford ourselves selecting only one streak per light source/bright polygon, because we need to generate smooth streaks for large bright areas due to photorealism requirements. Moreover, due to object domain and application specifics, we cannot obtain position of the streak generating sources in any other way than analyzing per-pixel brightness of the final image.

3.2 Integrating Streaks into the HDR Rendering Pipeline

Another problem is how to combine streaks effect with other image-space effects like bloom, tone mapping and antialiasing.

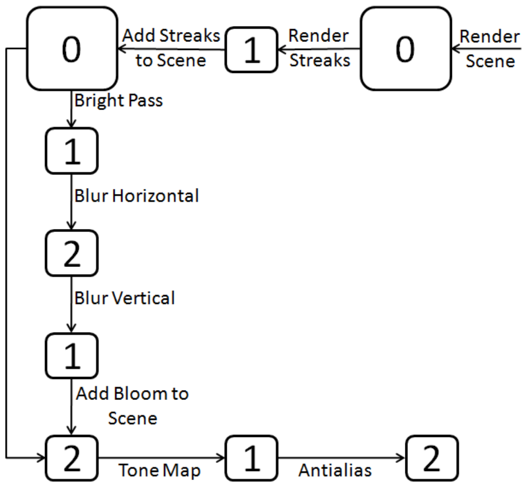

Figure 4: Rendering pipeline with streaks and bloom.

Taking into consideration that streak effect is linked with diffraction in eyelashes/lens (i.e. *ahead* of retina/film/matrix) and bloom effect occurs in the retina/film/matrix itself, it is clear that bloom should be added to an image after the streaks.

We use two rendering targets (color buffers) of screen size ('small') for ping-ponging [2] between them during blur rendering and one of double screen size ('large') for supersampling antialiasing [4]. As shown in Figure 4, we render streaks and bloom to the small buffers. Due to smooth nature of bloom and streaks, they don't require being antialiased, and rendering them into small buffers saves fill rate and rendering time.

We also introduce a new method of tone mapping [9], specifically adapted for our application area. Motivation and the algorithm itself are explained in Section 3.3.

After all the passes have been executed, adaptive shader blur takes place, which has been fine-tuned to give an image anti-aliased look, and only affects areas of the screen space, where rendered polygons adjoin [10].

3.3 Tone Mapping Algorithm

Since streaks are mapped on the image in HDR buffer before its conversion to low-range (8bit), they take part in tone-mapping step.

Tone mapping is needed because after rendering our screen buffer contains unbound intensity values which and needed to be converted into low dynamic range [0..1] and a specific RGB color space. The latter is important because after simulation we can get colors which cannot be represented by the gamut of the target (monitor) color space. So a tone mapping algorithms deals with two types of out-of-gamut values: high intensity (>1) value and non-RGB values (<0).

Other requirements for tone mapping were high (interactive) speed and preservation of color tone for high intensity pixel values. Note than we don't need to compress all dynamic range into low range, rather we need to simulate camera behavior, so in our application user can select desired exposure ratio.

Out tone mapping is based on simple global intensity scaling solution with some modifications. To match requirement of color tone preservation we scale all components of color proportionally in case there are at least one component with value greater than one. This allows keeping correct object color tone in contrast to typical approaches based on simple linear or logarithmic mapping, like in [1]) (see Figure 5).

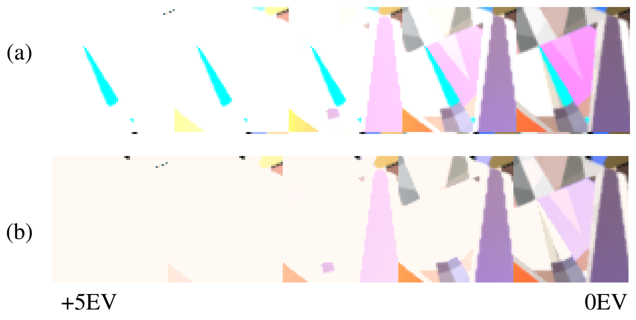

Figure 5: (a) wrong color behavior with increasing exposure on simple tone mapping (b) realistic behavior on our algorithm

For dealing with negative values, we transform rendered spectrum values into CIE XYZ 1931 color space which is guaranteed not to have negative values. Tone mapping is performed in this space and only after that we return to monitor RGB space with gamut mapping (clipping). Working in XYZ space helps us correctly tone-map out-of-gamut color and also to get more realistic result for saturated colors. Saturated colors are usually poses a problem for global tone mapping approaches because very bright pixels don't become white even with high exposure ratios (see Figure 5), which gives unnatural result.

Our tone mapping technique produces correct and predictable behavior on different exposures.

4. ACCELERATION TECHNIQUES

Though we have implemented all the described optimizations, streaks rendering time is still unacceptable. The bottleneck is in the GPU's fill rate, because generating streak for each point, we render a textured quad with additive blending (i.e. without z-buffering and z-cull).

Therefore we involve techniques limiting the number of screen-aligned textured quads that are actually drawn.

4.1 Clusterization

Consider an input image for the streaks rendering algorithm. Normally, bright points are joined in groups, each of the groups corresponding to a bright polygon or another light source.

Consider an N×N pixel region, where each pixel is bright enough to produce a streak. Let's precompute a streak (we call it 'integral streak'), assuming that each pixel has the same color (see Figure 6).

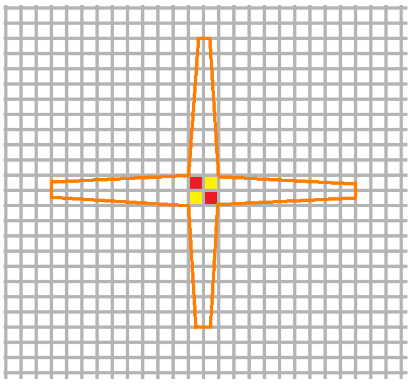
Figure 6: Integral streak for a 2×2 region (bright points are filled with red and yellow, resulting streak contour – with orange).

Let's divide an input image by a grid with a grain size of N×N. Some of the grains will be filled with pixels, all of which are bright enough to produce streaks. Assuming that their colors do not vary greatly (which is fairly probable), instead of rendering four 'single' streaks, we can render one 'integral' streak with the color, equal to the average color of these streaks. See Figure 7 for example.

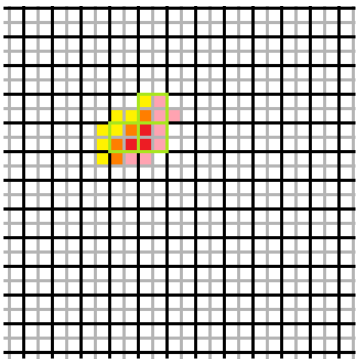
Figure 7: Applying integral streaks. Borders of candidates for integral streaks use are highlighted with green.

4.2 Quad-Tree Adaptive Clusterization

Selecting cluster size is a tricky problem. If we select it too small, some large streaks sources that could be optimized well will be optimized worse. If we select it too large, some small light sources will not be optimized at all.

We propose a method, which adaptively select the best possible size of the cluster for each image fragment. Assume that the maximum cluster size is 2^n. Then we need *n* integral streaks and one single streak image. First, we try to use the largest integral streak for a cluster. If it fails, we recursively divide it and repeat the process for the case of (n-1).

Since quad-streak clusterization renders more integral streaks than any 2^n clusterization, it provides the best speed (but the 'worst' quality, too).

In Figure 8 it can be seen that a 2×2 clusterization introduces 160% optimization, compared to the conventional rendering. This estimation is obtained comparing the total number of bright pixels (96) to the sum of 2×2 clusters and non-clusterized pixels (20 and 16). Using the similar calculations we can obtain that a 4×4 quad-tree one introduces another 20% optimization, compared to the 2×2, but that could be one of the worst cases.

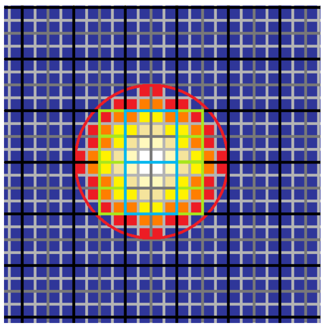
Figure 8: Using a 3-level quad-tree clusterization. Pixels that don't produce streaks are colored in blue. Light blue and light green shows the borders of the 4×4 and 2×2 clusters accordingly.

4.3 Comparison of Speed of Different Streak Rendering Techniques

We have tested the proposed methods on different image sets and measured time elapsed during streaks rendering. The results are shown in Figure 9. Notice the time difference between the no cluster, 2x2 static cluster and 16x16 quad-tree cluster methods.

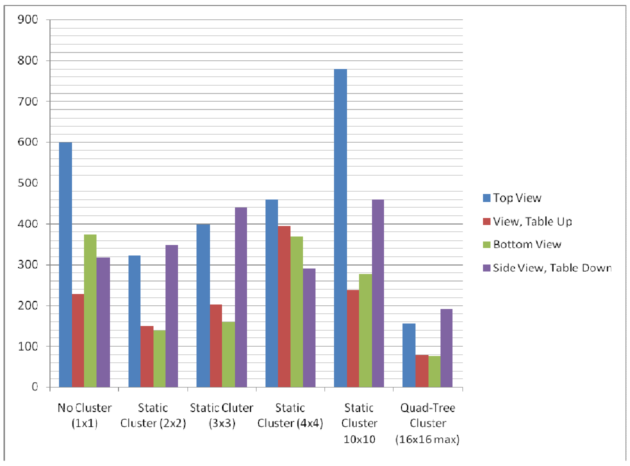

Figure 9: Comparison of streaks rendering speed for different cluster sizes. Time is given in milliseconds.

Test configuration: Dell Inspiron 1520, Intel Core 2 Duo T7500 (2.2 GHz), GeForce 8600 M GT, 2 Gb RAM

6. CONCLUSION AND FUTURE WORK

In this paper we propose a novel method for glare rendering. It offers a speed increase of up to four times, compared to the known physically-based method [1]. We've also made a research on the problem of correct integration of the streak effect into the rendering pipeline, and selected the fastest scheme available.

Future research should be made on both acceleration and quality enhancement. Clusterization could introduce a few new artifacts, and although we have corrected it, more R&D should be done on that subject. Besides, with the growth of GPU fill rate, other algorithm steps, like the search for bright points and recursive clusterization may become a bottleneck.

Although we have implemented these features on the CPU, they could be easily performed on a GPU, limiting bandwidth bottleneck (during image read from a framebuffer) and parallelizing image processing. Instruments that allow multiple arbitrary size output from a shader/thread, such as geometry shader or CUDA might be used.

7. REFERENCES

[1] Masanori Kakimoto, Kaoru Matsuoka, Tomoyuki Nishita, Takeshi Naemura, Hiroshi Harashima. *Glare Generation Based on Wave Optics. In Proc. of Pacific Graphics 2004, pp. 133-142, 2004, Seoul.*

[2] Chris Oat. *A Steerable Streak Filter. In ShaderX^3: Advanced Rendering with DirectX and OpenGL, edited by Wolfgang Engel, pp. 341-348, Charles River Media, 2004.*

[3] G. Spencer, P. Shirley, K. Zimmerman, D. P. Greenberg. *Physically-Based Glare Effects for Digital Images. In Proceedings of SIGGRAPH '95, Computer Graphics Proceedings, Annual Conference Series, Los Angeles, pp. 325–334.*

[4] Tom McReynolds, David Blythe. *Advanced Graphics Programming Using OpenGL. Morgan-Kaufmann, 2005.*

[5] Tristan Lorach. *Sparkling Effect, White Paper, available at http://developer.download.nvidia.com/whitepapers/2007/SDK10/Sparkles_hi.pdf.*

[6] E. Nakamae, K. Kaneda, T. Okamoto, T. Nishita. *A Lighting Model Aiming at Drive Simulation. In Proc. SIGGRAPH '90, August 1990, pp. 395–404.*

[7] Greg James and John O'Rorke. *Real-Time Glare. In GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Rendering, edited by Randima Fernando, pp. 343-361, Addison-Wesley Professional, 2004.*

[8] Tiago Sousa. *Adaptive Glare. In ShaderX^3: Advanced Rendering with DirectX and OpenGL, edited by Wolfgang Engel, pp.349-355, Charles River Media, 2004.*

[9] Larry Gritz, Eugene d'Eon. *The Importance of Being Linear. In GPU Gems 3, edited by Hubert Nguyen, pp. 529-542, Addison-Wesley Professional, 2007.*

[10] Oles Shishkovtsov. *Deferred Shading in S.T.A.L.K.E.R., In GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, edited by Matt Pharr, pp.143-166, Addison-Wesley Professional, 2005.*

About the Authors

Peter Sikachev is a 4th-year student at Computational Mathematics and Cybernetics department of Moscow State University. His research interests include photorealistic 3D rendering, interactive visualization and GPU programming. His contact e-mail is psikachev@graphics.cs.msu.ru.

Ilya Tisevich is a PhD student at Moscow State University, Department of Computational Mathematics and Cybernetics. His contact email is itisevich[a_t]graphics.cs.msu.ru.

Alexey Ignatenko is a researcher at Computational Mathematics and Cybernetics department of Moscow State University. His research interests include photorealistic 3D rendering, 3D modelling and reconstruction, image-based rendering and adjacent fields. His contact e-mail is ignatenko@graphics.cs.msu.ru.

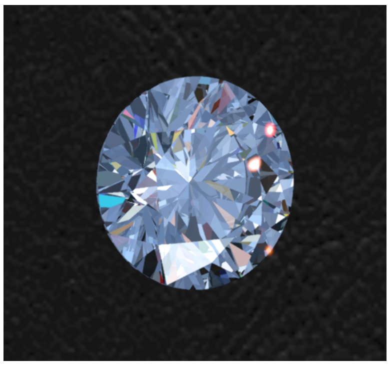
Figure 10: Bloom

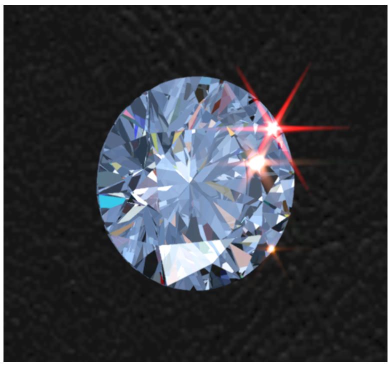
Figure 11: Streaks only.

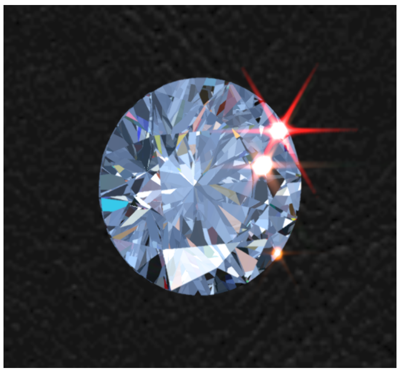
Figure 12: Streaks and bloom combined.

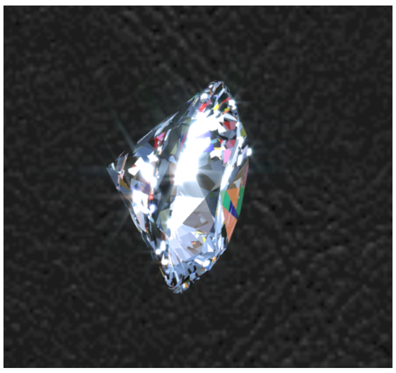
Figure 13: High quality streaks (cluster size 1).

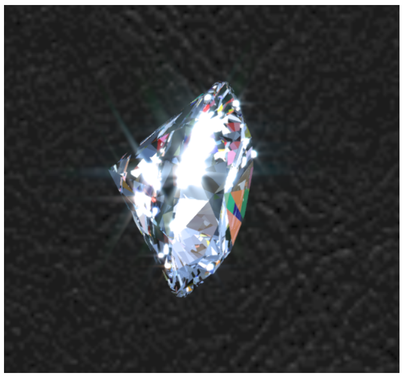
Figure 14: Low quality streaks (cluster size 2).

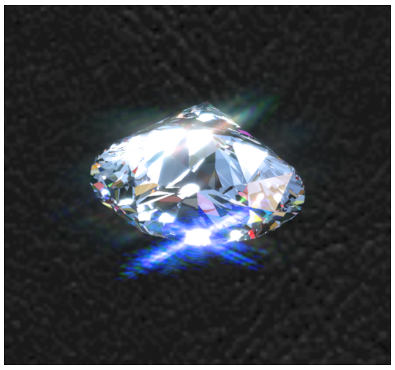
Figure 15: Streaks from eyelashes.

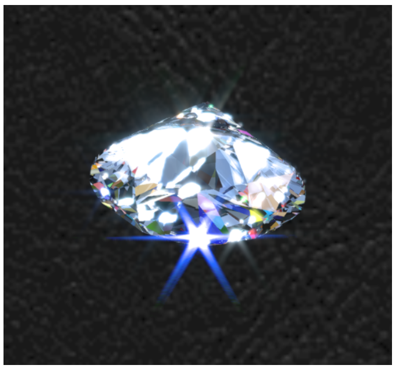
Figure 16: Streaks from camera diaphragm.