# Final Project Report

Yahong Zhu – 21333370

## Abstract

Lens flare effect is caused by light passing through a photographic lens system in an unintended way. It is often considered as artifacts in images, however, in modern computer graphic, it is a crucial component for creating photo-realistic imagery. Graphic artists generate it in images on purpose in order to enhance the scene and add a perceived brightness. In this final project, a method to interactively generate pseudo flare rendering for camera lenses is presented in real-time. The lens flare effects contain many components which are considered important for realism, such as ghosts, halos, chromatic distortion, lens dirt, diffraction starburst and blur effect. This method is not physically-based, however, it could be used as an alternative to it or an enhancement to the traditional sprite-based effect.

## Background

In the research paper, Hullin [2011] presented a novel method to compute physically plausible lens flare renderings by generating various important components in order to achieve a more realistic effect, for instants, imperfections, chromatic and geometric lens aberrations, and antireflective lens coatings. They also achieved various acceleration strategies to get a tradeoff between the performance of their system and the quality of the generated lens flare effects, which allows their system easily applied both in real-time application such as games as an enhancement of the scene and in high fidelity production rendering such as movie production.

Firstly, they described several aspects of the optical system to give a basic view of their transport model. The optical system includes lens design which represents the shape and size of the camera aperture and the light propagation inside the lenses. The Fresnel equations provide different transmission and reflection between different media. Then, Huygens' principle states the diffraction in this optical system. They encountered two types of diffraction effects: the starburst shape centered around the light source and the halos around the border of the reflection ghost. In order to get a computationally cheap approximation, they precomputed a small set of aperture textures by Fraunhofer and Fresnel approximations to the diffraction integral.

After modeling various important aspects of lens flare effect, they implemented their rendering system based on those physical principles. They are able to simulate the light propagating through the lens based on ray tracing techniques that are going through the optical system to the sensor. Instead of tracing all the light rays, they only choose a subset of rays to get the beam tracing approximately. In their system, they first compute the ghosts caused by the interreflection of each flare element which correspond to a specific sequence of transmissions and reflections. Then, they pick a subset of rays to track through the system, which achieves bundle tracing. Once the rays have been traced in the system, they conduct the rasterization and shading process in the system. A ray grid is formed on the sensor which is used after in the interpolation process to estimate the behavior of an entire ray beam. In this stage, they are able to render ghost shape and "Fresnel-like" diffraction patterns based on various aperture texture.

Finally, they present an acceleration method to improve the system. They achieve ray bounding by restricting the sparse set of rays to a rectangular region on the entrance aperture. In order to decrease the high frequencies in the optical system, a regular grid of incident rays is mapped to a homogeneous grid on the sensor. And then, they adapt the grid resolution for each ghost. They control the intensity of lights by fixing the number of the ghosts to control the budget. Furthermore, symmetries in the optical system can help reduce computational complexity. In the end, spectral rendering is applied to deliver a blended smooth result.

## Implementation

Inspired by the paper above, I implemented a simple lens flare effect in my final project. Similar components are achieved and presented in this project such as various ghosts, ringing and starburst patterns. The aperture that is used to generate basic ghosts shape is image based which mean depending on different images or environment textures we applied, the final effect varies. Because of the time limited, instead of using physically based techniques which obtain much better effect, we implement a pseudo lens flare that is easier to achieve with satisfied result.

There are four stages in our algorithm to generate a relatively realistic lens flare result, that are down sample and threshold, features generation, blur effect and upscale and blending

In the first stage, we down sample our source image in order to reducing the cost of following stages. Then, we select a subset of the brightest pixels in the source image to participate in our lens flare effect. It could be easily achieved in the fragment shader by the sampleScaledBiasedTexture and textureDistorted functions. After that, the input pixel would be dynamically selected form the image.

In the second stage, we generate several flare features based on the pixels we get from the previous stage. All the features of lens flare tend to appear around the center of the image, so we first flip the result from previous stage by flipping the texture coordinates in the fragment shader. Then, we are able to compute the ghosts, halos and chromatic distortion in our shader. Ghosts are the blobs appeared in the inner part of the flare. We create several uniforms to control the properties of the ghost including the number of the ghost and the dispersion factor. We use fract() to ensure that the texture coordinates wrap around and give a linear falloff from the image center to get a weight samples. Those are operating in a loop to sample the textures. Finally, we give our ghosts the color by a uniform.

Halos are generated similarly to ghosts, we take a vector to the center of the image and fix the vector length, the radius of halos is controlled by haloWidth uniform. The last feature is the chromatic distortion which is caused by refraction of the light. We could simulate this by textureDistorted function which gets the red, green, blue channels separately. In the end, we added our features together to obtain a combined result.

In the third stage, we applied blur effect to our lens flare features to reduce the coherence with the source image. Without adding it, the lens flare features retain the appearance of the source image which is not exactly what we want. We use gaussian blur in our fragment shader to achieve that result.

In the last stage, we blend the lens flare effect with our source image, however, before that there are several things we could add in the flare to enhance the final result. First, we add lens dirt (Fig.1) to make our flare effect more realistic. This is applied by adding a full-resolution texture that simulates the dirt on the camera lens. That effect gives us some imperfections on the flare which presents a unique aesthetic appearance. Second, we could add a starburst texture (Fig.1) in addition to the lens dirt for a further enhancement. Since the starburst texture is static texture as well, we use a transformation matrix to rotate it in every frame to produce a dynamic result. After producing all these additional features, we multiplied them together to get a blended effect in our scene in the fragment shader.

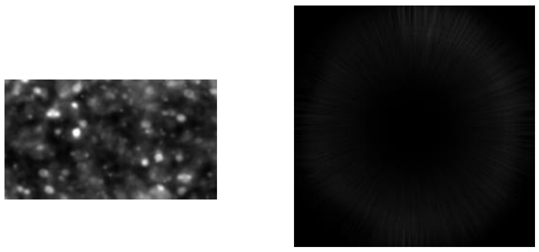


Figure 1 Lens dirt textures

## Results

The project is implemented in GLSL and using C++ in Visual Studio 2017. The approach we used reaches a real-time lens flare effect interactively rendered based on the source image we choose. The result is shown below (Fig.2) with original image and the image blended with our rendered lens flare. The algorithm produces lens flare rendering with convincing lens flare features generated in real-time.

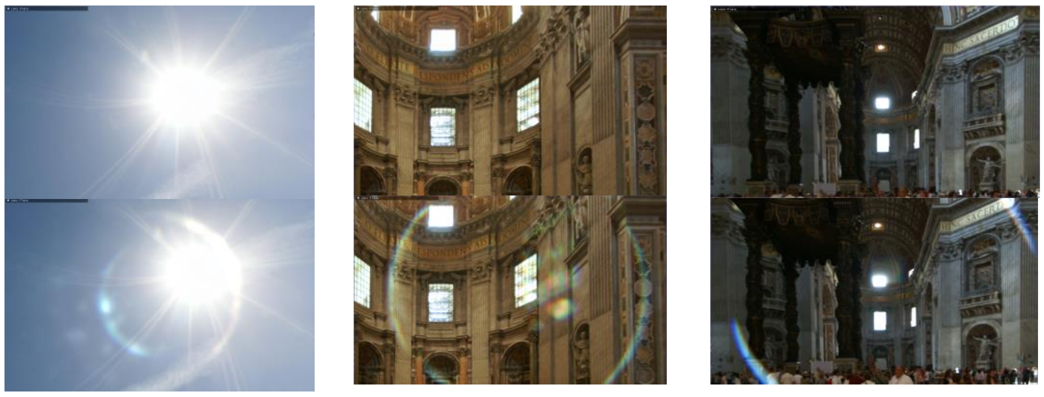
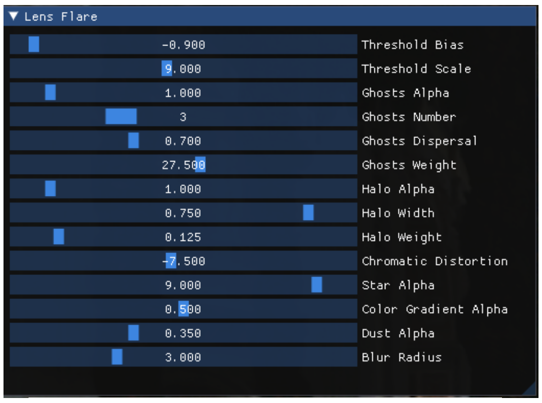


Figure 2 Results of generated lens flare with different source images

While moving the source image, it can be seen that the ghosts are generated by the brightest set of pixels appeared in the center of the scene. The rendered effect improves the original images and gives special aesthetic result which proves our approach simulates these natural artifacts convincingly.

In order to discover various attributes in this lens flare system, we introduce imGui as the user interface widely used in OpenGL. A list of attributes is shown on the scene, we change different values to modify the shape and the size of our lens flare to adjust the source image and create better visual effects. The list could be seen on Fig.3 below.



▼ Lens Flare

| | |
|---|---|
| -0.900 | Threshold Bias |
| 9.000 | Threshold Scale |
| 1.000 | Ghosts Alpha |
| 3 | Ghosts Number |
| 0.700 | Ghosts Dispersal |
| 27.500 | Ghosts Weight |
| 1.000 | Halo Alpha |
| 0.750 | Halo Width |
| 0.125 | Halo Weight |
| -7.500 | Chromatic Distortion |
| 9.000 | Star Alpha |
| 0.500 | Color Gradient Alpha |
| 0.350 | Dust Alpha |
| 3.000 | Blur Radius |

Figure 3 The list of lens flare attributes

## Limitations

There are couple of things that could be improved in the future for our rendering algorithm. First, in order to enhance it to get the realistic results, more complex algorithm could be applied in this system such as ray tracing introduced in the research paper.

The second one is the light sources in the scene. The current approach is only able to capture single light source in the scene, however, in reality multiple sources could be found in one scene. It is necessary to consider the influence of other source instead of just the center one so that we could achieve an accurate simulation.

The third one is that now we use image to generate the effect, but in practice we are more likely to embed the effect in a 3D scene which could be achieve by 3D models instead of 2D textures similar to skybox. That allows us to easily introduce our system into a game production operating in real-time.

The last limitation in our system is that there is no acceleration algorithm in our approach except down sampling in the first stage. We didn't measure the computation cost even though the effect could be generated in real-time. Yet if we embed this into a larger application, computational cost could not be ignored.

## References

i. M. Hullin, E. Eisemann, H.-P. Seidel, and S. Lee, "Physically-based real-time lens flare rendering," ACM Transactions on Graphics, vol. 30, no. 4, p. 1, Jul. 2011, doi: 10.1145/2010324.1965003.

ii. John-chapman-graphics: http://john-chapman-graphics.blogspot.com/2013/02/pseudo-lens-flare.html

iii. ThinMatrix OpenGL Tutorial 53: https://www.youtube.com/watch?v=OiMRdkhvwqg&t=199s

iv. Skybox texture downloaded from: http://www.humus.name/index.php?page=Textures