MVA2013 IAPR International Conference on Machine Vision Applications, May 20-23, 2013, Kyoto, JAPAN

Ray-Tracing Based Interactive Camera Simulation

Damon Shing-Min Liu
Computer Science Dept
National Chung Cheng University, Taiwan
damon@computer.org

Che-Wei Hsu
Computer Science Dept
National Chung Cheng University, Taiwan
a09007061@gmail.com

Abstract

Camera simulation aims to enhance realistic rendering, lens design and augmented reality by accurately simulating geometrical optics. Here our work focuses on optical phenomena, such as depth of field, monochromatic aberration, distortion, and aperture exposure, that are based on real camera lens architecture. Our approach is modeling pixel equation using ray tracing algorithm to render scenes after calculating the effect of lens refraction. We also provided an interactive user interface to control camera parameters, allowing for visual display of the corresponding camera lens system and of synthetic rendering result. To promote the performance of ray tracing algorithm, we improved our system using GPGPU programming language, CUDA, and OptiX ray tracing engine that is capable of parallel processing massive multi-ray sampling. We further optimized the sampling method for real-time pixel pupil calculation.

1. Introduction

As the complexity of virtual environment increases, it becomes a critical issue to create realistic images that may involve light simulation techniques such as camera model. Our research endeavor focuses on multi-lens system simulation. The evolvement of the camera lens module was developed from the simple pinhole camera to exploit the ideal lens module for achieving image with focal point.

In a real world camera, there are groups of monolithic or combined lens in camera lens system. They characterized lens system, such as wide-angle and fisheye, using different lens combinations. Although multi-lens specification can be calculated and displayed using approximate mathematical formula, we made it possible to visualize the process of designing or tuning lens using refraction of geometrics. Moreover, our approach can be used in a rendering process to actuate realistic image such as depth of field, monochromatic aberration, distortion, and aperture exposure.

We completed a multi-lens camera simulation using a ray tracing algorithm. To accelerate performance of the program, we utilized GPU and employed NVIDIA OptiX ray tracing engine to compute ray launched in parallel because ray tracing algorithm can be parallelized naturally.

2. Related Work

Pinhole camera model or thin lens model that is used to describe ideal and perfect lens in image formatting process. Most traditional computer graphics images are based on pinhole camera to synthesize scene image because pinhole camera is easy to be implemented [1]. Thin lens model implemented the depth of field and motion blurs in Potmesil's [2] and Cook's [3] research in the 1980s. Zhou et al. [4] introduced a two-pass filter that blurs the sampling pixel by coloring from neighboring pixels. Michael et al. [5] proposed using heat diffusion to simulate blurring of depth of field. They implemented the depth of field with thin lens camera model using the image-based post-processing method.

In thick lens system, a lens is composed of two surfaces with related positions of focal point and principal point. Multi-lens camera lens model consists of complex lens system that can be replaced by a black box in paraxial geometrical optics system. To combine that focal distance of separated lenses, the entire focal point can be calculated using Newtown's formula [6]. Kolb et al. [7] implemented the multi-lens camera model and accurately simulated the behavior of physical systems containing an aperture and camera system architecture for a geometrically and radiometrically correct camera model. The completed full lens system includes optical and lens fundamentals such as diffraction, material coatings, and lens flare effects. Steinert et al. [8] and Hullin et al. [9] implemented optical realism synthetic image using ray tracing algorithm. The flare and diffraction were demonstrated in lens effect by shedding lights pass through the complex lenses.

Our research is based on Kolb's work and focuses on the geometry between lenses so that the advanced optical theory can be applied for simulating the camera system using ray tracing algorithm. Instead of using post-processing methods, we exploited ray tracing algorithm in our research for realism and characteristics of light path, but in it thin lens camera system still fails to model a number of optical phenomena and lacks for aperture to obtain the correct exposure.

The OptiX engine is a low level ray tracing engine functioning on highly parallel architectures of NVIDIA GPU. Engineers can program ray tracing pipeline using small set of operations. For easy programming the code of ray tracing kernel, OptiX, is based on CUDA, a C-extension programming language, rather than using SIMD-style constructs. It also provides just-in time compiler and object model acceleration structures to implement its programs more efficiently [10].

Geometrical optics does not describe wave-particle duality of light; instead, that light is in terms of rays to model the propagation in a homogeneous medium. It usually is used in computer graphics for describing light propagation, refraction and reflection linearly. Refraction can be described according to Snell's refraction law [6], but geometrical optics fails to account for optical effects such as diffraction. Even though geometrical optics cannot describe wave-particle duality physically, the

approximation method still can model feature of the diffraction. In lens design, the combination of lenses proposes to decrease aberrations [11]. Some special requests of camera lenses still have aberrations as the special effect. To consider the effect of light, we tried to simulate optical phenomena as possible as we can.

3. Approaches

The behavior of geometry camera can be characterized by launching ray from image plane, and then ray transmits in air of camera, passing through lenses and aperture, and finally shooting to the object world. The rendering result is solved by ray tracing algorithm with super-sampling and exposure function. In addition, we obtain the rendering result, we obtain the refraction data of camera simulation from Optix system and display resulting image using OpenGL.

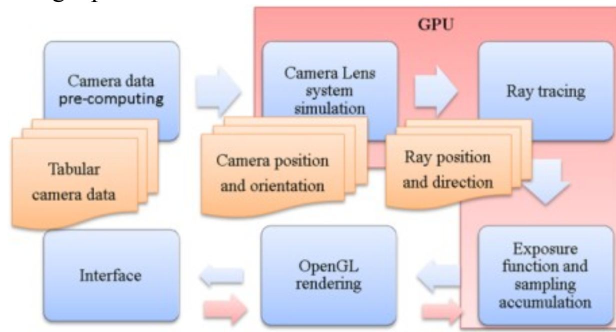
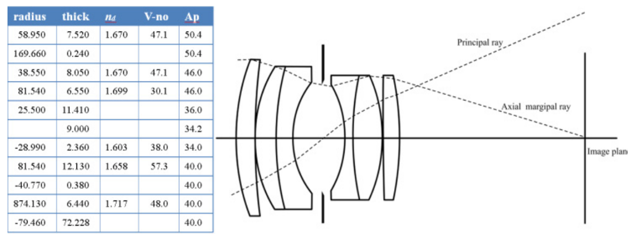


Figure 1. System overview.

3.1. System overview

We depict our system flow, as shown in Figure 1. The first step prepares coherent geometry camera data for camera simulation from tabular camera lens data. The second step runs in GPU to calculate ray and lens surfaces intersection as the unit of pixel with camera position and orientation and outputs ray's direction and position when rays pass through full camera lens system.



radius	thick	n_d	V-no	Ap
58.950	7.520	1.670	47.1	50.4
169.660	0.240			50.4
38.550	8.050	1.670	47.1	46.0
81.540	6.550	1.699	30.1	46.0
25.500	11.410			36.0
	9.000			34.2
-28.990	2.360	1.603	38.0	34.0
81.540	12.130	1.658	57.3	40.0
-40.770	0.380			40.0
874.130	6.440	1.717	48.0	40.0
-79.460	72.228			40.0

Figure 2. Lens surface data of Double-Gauss.

3.2. Camera simulation

In lens design book, tabular description of lens surface and aperture suffices to profile the view of a lens system [12]. Every lens or aperture can be reverted its geometry of surface or aperture as shown in Figure 2. We defined geometric lens surface using tabular lens data for a sequential list of spheres that were arranged along with axis by their center points to curvature lens surface. We pre-computed the geometric data of each lens surface, such as lens cap top, and center of sphere of lens surface, to characterize full camera system and used it for light path calculation, as shown in Figure 3.

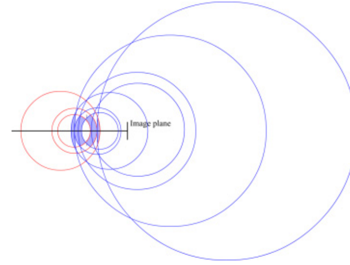


Figure 3. 2D sketch of lens spheres of Double-Gauss. Red spheres are negative radius of lens surface. Blue spheres are positive radius of lens surface.

To construct camera system, we utilized camera coordinate system to define camera position and orientation. Line equation in vector form is used to describe light path. We exploited ray-sphere intersection algorithm [13] and Snell's law to calculate the refraction on each sphere surface. We defined our image plane using a fictitious plane outside of the last lens surface and center of film, $\vec{c}$. With given film size that was defined by user, each pixel $p(x,y)$ in image plane can be calculated using camera coordinate system vector $[\vec{u} \quad \vec{v} \quad \vec{w}]$ as:

$$p(x,y) = \vec{c} + \text{Filmsize}/2 * (x\vec{u} + y\vec{v}) \tag{1}$$

In each pixel of image, we assume that a pixel receives different directional light paths from lens surface. To balance sampling quality, we randomly select light from lens surface and limit light inside of lens aperture. Every lens sphere can be defined as a set of points S with the radius of sphere Sr as in Equation (2) and Equation (3). To solve the point of ray-sphere intersection on S, we substituted line equation in Equation (3) and got the algebraic Equation (4).

$$S = [x_s \quad y_s \quad z_s], \vec{c} = [x_c \quad y_c \quad z_c], \tag{2}$$

$$(x_s - x_c)^2 + (y_s - y_c)^2 + (z_s - z_c)^2 = Sr^2 \tag{3}$$

We solved Equation (4) as the quadratic equation and verified that the roots of t should be a real number using the discriminant of the quadratic equation, and verified ray-sphere intersection point with solved roots.

$$\begin{cases} p_0 = [x_0 \quad y_0 \quad z_0], \vec{d} = [x_d \quad y_d \quad z_d] \\ x_s = x_0 + tx_d, y_s = y_0 + ty_d, z_s = z_0 + tz_d \\ (x_0 - x_s)^2 + (y_0 - y_s)^2 + (z_0 - z_s)^2 = Sr^2 \end{cases} \tag{4}$$

where $\overrightarrow{p_0}$ is start point, $\vec{d}$ is the line direction, t is the scale. As the vector form of line equation, we determined the refraction calculation according to Equation (5), where $\vec{N}$ is the unit normal of ray-sphere intersection point that was obtained from intersection point to sphere center; $\overrightarrow{R_0}$ is the direction vector of incident ray; $\overrightarrow{R_1}$ is the direction vector of refracted ray; n_1, n_2 are refractive indices. We assumed the aperture is a virtual circle in 3D whose circle center is on $\vec{w}$ and orthogonal to $\vec{w}$. When rays intersected with the virtual aperture, we did not calculate the refraction; if rays did not intersect with virtual aperture, they would be blocked.

$$\begin{cases} \overrightarrow{R_1} = \frac{n_1}{n_2}\overrightarrow{R_0} + \Gamma\vec{N}, \\ \Gamma = \sqrt{1 - \frac{n_1}{n_2}\left(1 - \left(\overrightarrow{R_0} \cdot \vec{N}\right)^2\right)} - \frac{n_1}{n_2}\overrightarrow{R_0} \cdot \vec{N}, \end{cases} \tag{5}$$

3.3. Scene ray tracing

Our ray tracing algorithm is implemented using OptiX

ray tracing engine, including ray launch, traversal, and shading in OptiX framework. In ray launch, the most important part is to implement ray generation program. Ray generation program weights the most in our work.

Ray generation program is parallelly executed in each thread [14]. Each thread is distinguished with a unique *rtLunchindex* that represents pixel of rendering result and start point for ray tracing.

Before ray tracing scene, we simulated rays launch from image plane passing through lens system using camera simulation method. If rays do not pass the lens system, it means ray was blocked by aperture or lens barrel, we assign it a background color, usually is black, for weighing accumulation result. To obtain the position and direction that we emitted from last lens surface, OptiX offers *rtTrace* API to trace and shade scene color.

For optimizing quality of resulting image, we exploited pixel pupil that was improved from exit pupil by Steinert et al. in [8]. They illustrated that different pixel locations are not imaged with full exit pupil because the effective aperture of vignetting. For interactive control, we implemented adjustable sampling method that tests passive sampling ray in each iteration of pixel ray tracing thread. We consider the passive sampling rays that pass through lens system and record their 3D coordinate to compute average center of position gravity and average distributed range for calculating the radius of distributed range in each pixel. Pixel pupil center and radius will be converged in sampling iterations when lens system is unchanged.

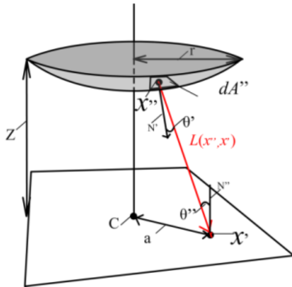


Figure 4. Geometry of the irradiance [7].

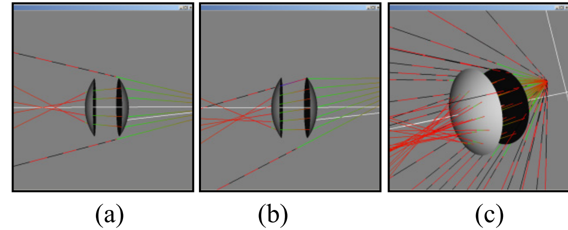
In camera simulation, exposure was introduced in Kolb et al.'s work [7]. Pixel irradiance can be written as in Equation (6). Exposure function corresponds to pixel irradiance from a small region on last lens surface *D*, with a during of shutter open *T*, as shown in Figure 4, the exposure can be written as in Equation (7).

$$H(x')=E(x')T \quad (6)$$

$$E(x') = \int_{x'' \in D} L(x'',x')\frac{\cos\theta' \cos\theta''}{\|x''-x'\|^2}\,dA'' \quad (7)$$

We compute each solid angle over pixel normal and lens surface normal on x' and x''. To solve the integral of exposure function, we use ray tracing algorithm that shads color from object world. The x'' is easy to be found using ray sphere intersection with given x' and a random direction. θ' and θ'' on x' and x'' can also be calculated after refraction calculation to obtain normal vector of lens surface on x''. Finally, the term $\frac{\cos\theta' \cos\theta''}{\|x''-x'\|^2}\,dA''$ is used to weigh the shaded result of sampling ray and D is averaged by number of sampling rays to dA''. Finally, we ask users to define the exposure time as a gain value to compute Equation (6).

OptiX maintains a *traverser* in mesh bounding box traversal step. In shade state of OptiX, *closest hit program*, *any hit program*, and *miss hit program* shads object surface irradiance, shadow, and background. *Closest hit program* is executed whenever Optix finds the closest intersection between a ray and an object [15]. In *closest hit program*, we employ Phong shading algorithm with *material template library* illumination modes and cast shadow ray and second order ray to measure shading color.



(a) (b) (c)

Figure 5. Aberrations simulation. (a) Spherical aberration (b) Coma (c) Astigmatism.

4. Experiments

Our experiment exploited OptiX ray tracing engine using the NVIDIA CUDA GPU computing architecture. To combine with display of simulation result, we use OpenGL 3.1 and GLUT 3.7.6. For GUI design in OpenGL that was no complicated windows GUI environment, we utilize GLUI 2.35 to pass parameters from interface to our system. In GPU, we use NVIDIA Quadro 600 which has 96 cores for highly parallel computing performance. Our development environment uses Microsoft Visual Studio 2008 with NVIDIA CUDA 3.2 and NVIDIA OptiX Ray tracing engine 2.5.1 in OS Windows 7 32bit.

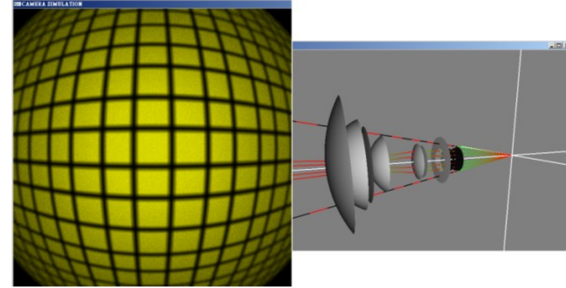


Figure 6. Image distortion and Fish-Eye camera.

First, we depicted light path in a single convex lens to illustrate simple aberrations as shown in Figure 5. We drew light path that was launched from a point with different directions. When lights pass through lens surface on places of different height, lights cause aberrations. As we sketch the light path, the rendering result of image distortion can be simulated in our rendering result window. We use a fish-eye camera model in [11] to emphasize distortion of scene as shown in Figure 6. The fish-eye camera shows barrel distortion, and the dark sides in corner are spaces that lens apertures occlude lights.

(a) Our result (b) OptiX example

Figure 7. Comparison of depth of field.

The other import phenomenon of light, depth of field, is simulated in our system using Double-Gauss lenses model that is shown in Figure 7. In close up scene, depth of field can be approximated using Equation (8).

$$\mathrm{DOF} \approx 2Nc\frac{m+1}{m^2} \quad (8)$$

where c is the diameter of circle of confusion; N is *f*-number; *m* is the magnification of lens. In Double-Gauss camera model, we assume coordinate of scene scales a unit to one pixel size that c can be assigned as 1mm. The depth of field can be calculated as 7.86 mm that was close to the value we observed in rendering result. OptiX provides a depth of field camera model that uses thin lens model, but it gets narrow depth of field and blur promptly.

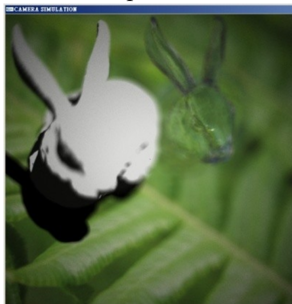


Figure 8. Rendering result.

System performance depends on number of pixels that needs to execute mesh traversal in ray tracing algorithm. In a thread, its work load per pixel, 94.9% of GPU execution time was spent in OptiX ray-tracing; 4.2% spent in camera lens refraction simulation using GPU. When complexity of scene increases, the ray tracing algorithm launches rays for multi-order reflection on object surface or refraction in translucent object that can bring per frame performance down. For one sampling ray in each pixel, our system performance is 1.537 frames per second that uses Double-Gauss camera, as shown in Figure 8, and sampling 20 times in 640 x 640 resolution.

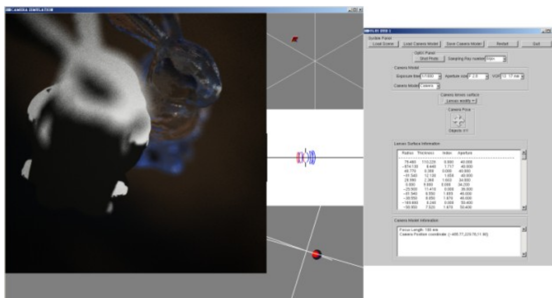


Figure 9. System overviews with GUI.

As shown in Figure 9, our GUI interface lists camera parameters as in a real camera system to shot scene. We support kinds of camera lenses surface data for user to acquaint with inside of camera. Moreover, our system can modify lens surface data to create unique camera model.

5. Conclusion and Future Work

We simulated geometric multi-lens based on ray tracing algorithm. This system demonstrates refraction using Snell's law. In it, light path can be accurately calculated for aberrations and depth of field. Ray tracing algorithm accompanied with parallel computing can enhance efficiency and performance in sampling method to curtail overhead of experiment. Camera simulation improves the realization of light characteristic and visualizes light transition path for observing and proposes new surmise.

We need to acquire more light theory, such as wave-particle duality of light, to investigate diffraction that may cause in camera system. To accurately represent a pixel color in film, we need to launch large amount of sampling ray when relative focal point of each lens surface is not on target. For depth of field effect, we need more sampling rays in blurred field, but fewer sampling ray can represent full light field because all lights are focused on a point.

In camera simulation, ray transmissions can be described in matrix form. Full camera lenses are treated as matrixes that can be pre-computed into single transmission matrix to decrease computing time. OptiX presents a general-purpose ray tracing algorithm framework. To optimize ray tracing algorithm and reduce stumbles, pure CUDA programming may speed up.

References

[1] P. Lönroth, et al.: "Advanced real-time post-processing using GPGPU techniques," *DICE*, 2009.

[2] M. Potmesil, et al.: "A lens and aperture camera model for synthetic image generation," *ACM SIGGRAPH*, vol. 15, no. 3, pp. 297-305, August 1981.

[3] R. L. Cook, et al.: "Distributed ray tracing," *ACM SIGGRAPH*, vol. 18, no. 3, pp. 137–145, July 1984.

[4] T. Zhou, et al.: "Accurate depth of field simulation in real time," *Computer Graph Forum*, vol. 26, no.1, pp. 15-23, March 2007.

[5] M. Kass, et al.: "Interactive depth of field using simulated diffusion on a GPU," *Tech. Rep., Pixar Animation Studios*, no. 06-01, 2006.

[6] M. Katz: "Introduction to geometrical optics," *World Scientific*, pp. 36-103, 2002.

[7] C Kolb, et al.: "A realistic camera model for computer graphics" *ACM SIGGRAPH*, pp. 317–324, 1995.

[8] B. Steinert, et al.: "General spectral camera lens simulation," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1643-1654, March 2011.

[9] M. Hullin, et al.: "Physically-based real-time lens flare rendering," *ACM Transactions on Graphics*, vol. 30, no. 4, pp.108:1–108:9, July 2011.

[10] B. Barsky, et al.: "Camera models and optical systems used in computer graphics: Parts i and ii," *ICCSA*, pp. 246–265, 2003.

[11] W. Smith: Modern Optical Engineering, *McGraw-Hill*, 3rd edition, pp. 61-89, 2000.

[12] W. Smith: Modern Lens Design, *McGraw-Hill*, 2nd edition, pp. 85-88, 2005.

[13] Ray-Sphere Intersection, http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtinter1.htm

[14] "OptiX Ray Tracing Engine Programming Guide version 2.1," *NVIDIA CORPORATION*, pp. 30-47, September 2009.

[15] "OptiX Ray Tracing Engine Quickstart Guide version 2.1," *NVIDIA CORPORATION*, pp.18-24, September 2010.