Simulation of Camera Features

Michal Kučiš*
Supervised by: Pavel Zemčík†

Faculty of Information Technology
Brno University of Technology
Brno / Czech republic

Abstract

Computer vision algorithms typically process real world image data acquired by cameras or video cameras. Such image data suffer from imperfections caused by the acquisition process. This paper focuses on simulation of the acquisition process in order to enable rendering of images based on a 3D generated model that are as close to the images acquired by cameras or video cameras as possible. Its main purpose is simulation of video input for computer vision applications, e.g. robot navigation. Imperfections, such as geometry distortion, chromatic aberration, depth of field effect, motion blur, exposure automation, vignetting, inner lens reflections, and also imperfections caused by image sensor features are considered.

The paper, besides description of imperfections of the acquisition process and description of imperfections simulation, also presents results of the simulation software through illustrative figures produced by the software.

Keywords: Camera Imperfections Simulation, Depth of Field Effect, Distortion, Motion Blur, Vignetting, Lens Flare, Image Sensor Features

1 Introduction

Computer vision is a field of computer graphics that allows for acquiring, analyzing, and understanding images. An input of computer vision algorithm is a set of images, which are typically captured by camera or video camera. An output is a symbolic information, e.g. information about identity of subject in the image or any other valuable information. Input images are not perfect copies of the reality since they are affected by many camera features. A "perfect" image of the real world is modified and camera features add imperfections to the image. The imperfections strongly influence success of the algorithms, so in general, the algorithms need to take into account these imperfections.

Implementations of computer vision algorithms in real world application work with images or sequences of images, which are usually captured by a camera. Such can be e.g. a security camera or a robot camera. Also, the implementations have to be trained and tested on a set of images. The more similar the test images and the images captured in the real world application are, the more precise the results of algorithms are.

The best way to acquire similar data is to use a camera, which would be used in the real world application. However, this option is often either expensive or unavailable. The second option is to use a dataset of images, which was acquired by a different camera or cameras. The images would be affected by different camera imperfections, so the result would be worse. The third option is to generate data by computer and use it as input images. The generated images and dataset of images are generally cheaper, but the results of such solution would be worse as well.

A good way how to get better results is to modify computer generated "perfect" images to look similar to the images created by a target device. This paper describes a simulation method for acquiring target device like images.

2 Related work

No complex simulator with all the desired features was known up to the date, but there are relatively many publications that describe the camera features and simulation methods for some of the features [4, 5, 8, 10].

Distortion is a general problem of lenses. Measurement and correction of the distortion are described in [12]. Simulation of distortion is just a reversed process. Detection and elimination of a chromatic abberation (a form of distortion) is shown in [5].

Depth of field effect problem is widely explored. Many algorithms that simulate the effect are known. Basic description of these methods can be found in [3, 6, 10]. Detailed description of the interactive depth of field diffusion method is described in [2]. This method is interesting because it does not suffer from some problems that appear in the other post-processing methods.

Many camera sensors capture color images using a color filter array. Description of this feature and its effect to the output is described in [11]. Another problem of the sensors is noise. Noise evaluation of CCD sensors is shown in [9].

*xkucis00@stud.fit.vutbr.cz
†zemcik@fit.vutbr.cz

Characteristics of many cameras and sensors are described by EMVA Standard 1288 [1].

3 Description of simulation

The proposed algorithm consists of application of several effects, which simulate the described features. All the simulated effects are described in the following sections and every one can be adjusted by using parameters. Default order of the effects application is shown in Figure 2. (In the implemented software, the parameters of the effects and their order are described with configuration file.) Input of the simulator is a (high-dynamic range) color image, a depth map, and an environment map. The color image is a "perfect" image that will be modified by the simulator. The image is shown in Figure 1. The depth map defines distance between a pixel in real world and a camera. This map is used by the depth of field effect and the motion blur effect.

Figure 1: Input color image; this image is modified by the simulator

4 Lens features

This section describes lens features: distortion, chromatic abberation, vignetting, and lens flares. The camera lens is an optical system that affects more phenomena like spherical aberration, astigmatism, coma and more, but in general their influence in the camera lens is not considered or it is rarely considered. Therefore, we have not included these phenomena.

4.1 Distortion

The camera image is a 2D-projection of the real world. In optics, distortion is a deviation from rectlinear projection. This projection guarantees straight lines remain straight after the projection.

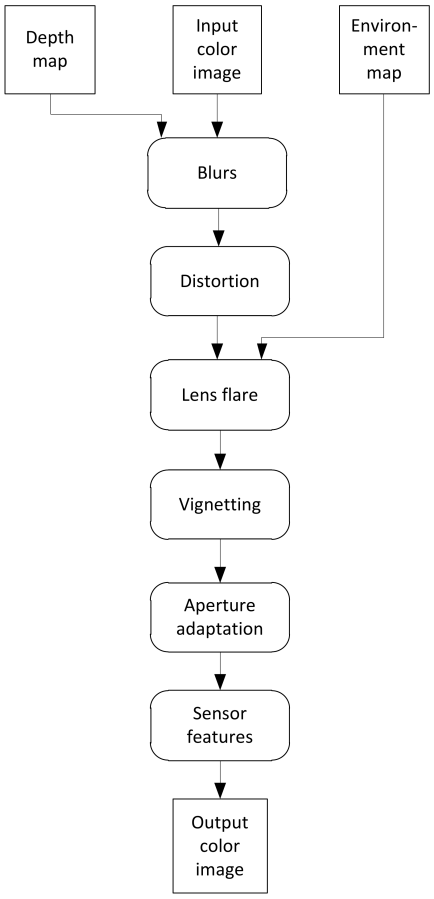

Figure 2: Algorithm of the simulation. The effect 'Blurs' links up depth of field effect, motion blur, and structural blur. The effect 'Sensor features' links up noise and color filter array.

The most commonly encountered distortion is radially symmetric distortion. This type of distortion arises from the symmetry of the camera lens and the symmetry of the camera optical system. In this case, the distortion can be simulated by the radial distortion model. The radial distortion model uses 6-7 real-valued variables which specify the distortion. [12] contains a description of this model.

Radial distortion model can be expressed as:

$$r_d = f(r), \tag{1}$$

where: r_d – destination distance of input pixel position
r – source distance
f – distortion function

If the distortion is not symmetric, it cannot be simulated by the radial distortion model. In that case, warping is used that as described in [7, 14].

The distortion simulation uses a 2D filter to remap the input pixels to the output pixels. For better results, we use the Lanczos filter in the current version.

4.2 Chromatic abberation

Chromatic abberation is a type of distortion in which the lens is not able to focus all colors to the same convergence point. This distortion is caused by different refractive indexes for different light wavelengths. The distortion is mainly observed in the edges between a bright light and a shadow.

The abberation can be simulated by modification of the distortion model. Every color channel of the input image is deformed by a slightly different distortion. The result is a color contour in the edge of the bright and dark areas. The abberation is radially symmetric. Therefore, it can be implemented by radial distortion model and also by warping.

We simulate the abberation by:

$$r'_{color} = f_{color}(r), \tag{2}$$

where: r'_{color} – distance between the output pixel position and a center of the image
r – distance between the input pixel and the center
$f_{color}(r)$ – central distortion model for $color$

4.3 Vignetting

Vignetting is a reduction of an image's brightness at the image periphery. The vignetting is mainly radially symmetrical, but rarely it can be also radially asymmetrical. The simulation of this feature is straight-forward. Pixels of the input image are multiplied by the pixels of a vignetting mask image. The vignetting mask is defined by the used camera and can be obtained by measuring. It can be measured in a scene where only one solid color occurs (e.g. a white paper). In this case, an image brightness is the vignetting mask. The example of the vignetting is shown in Figure 3.

Vignetting is calculated as:

$$B(x,y) = m(x,y)S(x,y), \tag{3}$$

where: B – output image
m – vignetting mask
S – input image

If it is centrally symetric vignetting, m is computed:

$$m(x,y) = m'\left(4\frac{(x-sx/2)^2+(y-sy/2)^2}{sx^2+sy^2}\right), \tag{4}$$

where: (sx,sy) – image size
m – vignetting mask

4.4 Lens flare

Lens flare is an unwanted light in lens system caused by inhomogeneities in the lens. The source of the lens flare are

Figure 3: Vignetting; the corners are darker than the center

unwanted internal reflections or scattered reflection inside the lens. It is difficult to describe all phenomena and their effects to the output image due to complex construction of the lens. Every lens creates different artifacts. Even the same type of the lens can create different artifacts under different conditions. In addition, the different position of the light source in the image can create different artifacts as well.

Lens flare manifests itself as a haze across the image or as visible artifacts. The haze can be simply simulated by adding color to all pixels in the image. The visible artifacts can be caused by a reflection on the aperture, inner reflections in the camera lens, refraction on inhomogeneities in the lens etc. Bright light source can create a star or can be mirrored etc.

A simple simulation of the lens flare artifacts is shown in Figure 4. We simulate this effect in following way:

$$B = S * K, \tag{5}$$

where: B – output image
S – input image
$*$ – convolution
K – convolution kernel

In Figure 4, a mirrored ghost is shown in the red circle. The ghost is created by:

$$B(x,y) = S(x,y) + \alpha_{xy}S(sx-x,sy-y), \tag{6}$$

where: B – output image
S – input image
α_{xy} – intensity of the ghost
(sx,sy) – size of the image

5 Aperture features

This section describes the simulated aperture features that affects the output image. It describes mechanisms of the

Figure 4: Lens flare effect; a bright light causes lens flare, the mirrored ghost of the light is shown in the circle

aperture adaptation and depth of field effects. This section also describes structural blur. This imperfection is not an aperture feature, but it can be simply simulated by a modified depth of field effect.

5.1 Aperture adaptation

Image brightness is influenced by three parameters in the real world application: an exposure time, a film speed and a f-number. The exposure time affects the motion blur. The film speed affects the sensor noise. The f-number describes a radius of the aperture. Moreover, the f-number affects the depth of field effect. With growing f-number, the depth of field effect becomes more visible (we suppose the f-number in form $f/1.2$, $f/2$, etc.). To calculate the output pixel color, we use these parameters in the following equation:

$$y_{ij} = x_{ij}\, c\, \frac{t\, s}{b^2}, \tag{7}$$

where: y_{ij} – output pixel color
x_{ij} – input pixel color
c – constant
t – exposure time
s – ISO film speed
b – f-number

The aperture adaptation calculates the f-number to get equal average grayscale value of image and middle gray. Middle gray is the universal measurement standard in photographic cameras and it stands a tone that is about half way between black and white. We compute an average image brightness by:

$$y_{avg} = \frac{\sum_i \sum_j w_{ij}\, x_{ij}}{\sum_i \sum_j w_{ij}}, \tag{8}$$

where: y_{avg} – average color
x – input image
w_{ij} – weight of the pixel

In general, pixels of our interest, e.g. in the center of the image, have bigger weight than the other pixels.

We implement two ways of adaptations. First one is in form:

$$b_{new} = \sqrt{y_{avg}}, \tag{9}$$

where: b_{new} – f-number
y_{avg} – average color of the image

The second method works iteratively. The drawback is, that the whole simulation has to be performed multiple times to get a valid f-number. On the other hand, the method usually converges to the true f-number. In addition, this method allows separate computation of average color and adaptation. The average color can be computed from the simulation input or the simulation output. This method is performed via a PID controller. The controller calculates an error value as the difference between the average color and the middle gray and it adapts the f-number to minimize the error. The formula is:

$$b_n = b_{n-1} + Kp\sqrt{e_n} + Ki \sum_{j=0}^{n-1} \sqrt{e_j}, \tag{10}$$

where: b_n – n-th f-number in the iterative computation
Kp – proportional gain
Ki – integral gain
e_n – difference between the middle gray and the average color with f-number equal b_n

5.2 Depth of field effect

Computer graphics methods for 3D scenes rendering typically use a pinhole camera model. The model leads to rendering entire scene in perfect focus. An image in the real world application is formed in an optical system where the light from a point in the scene converges at only one depth behind the lens. This depth is not necessarily equal to the sensor depth (Figure 5). The point in the real world appears spread over a region in the image. The region is called the circle of confusion (CoC). A computation of the circle of confusion is described in [3]. Simulation techniques of depth of field effect can be divided into object space methods and image space methods. The object space methods operate on a 3D scene representation. In general, the object space methods create more realistic results than image-space methods. The image-space methods operate on a 2D image of the scene. The image-space methods are postprocessing filters. The image is blurred with the aid of a depth map.

The simulator processes only 2D images, thus we are just interested in the image-space methods. We use two methods. The first method is based on the 2D linear filter

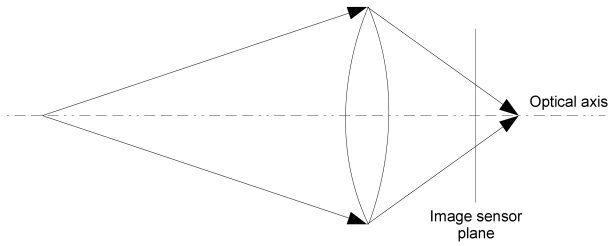

Figure 5: A point in the scene is projected as a disc on the image sensor plane, leading to depth of field effects

and it is called "Reverse-mapped Z-buffer depth of field". This method is shown in [6]. We use:

$$B(x,y) = \sum_i \sum_j psf(x,y,i,j)S(i,j), \tag{11}$$

where: B – output image
S – input image
psf – point spread function

The psf function is a point spread function, which relies on the circle of confusion computation.

The second implemented method is the depth of field using simulated diffusion (Figure 6). This method is based on principles of heat diffusion in an anisotropic environment. The algorithm is performed in two separate phases. In first phase, the diffusion is performed for each separate line. We create following equation for every pixel in the line:

$$y_i - x_i = \beta_{i+1}(y_{i+1} - y_i) + \beta_i(y_{y-1} - y_i), \tag{12}$$

$$\beta_i = min(CoC_{i-1}^2, CoC_i^2), \tag{13}$$

where: CoC_i – circle of confusion for i-th pixel
x_i – i-th input pixel
y_i – i-th output pixel

We get system of the equations that describes diffusion in the line. Then, we compute diffusion for every line of the image. In second phase, we compute diffusion for each column in the same way. This method is described in [2].

5.3 Structural blur

A perfect lens is able to project a point in the real world to an image point. A real world lens is not able to focus a real world point into an image point; therefore, the image of the point is blurred. We call this phenomenon the structural blur. We simulate this feature by modification of the depth of field effect computation. We just substitute circle of confusion computation by a structural blur ratio. Output of the structural blur is shown in Figure 7.

Figure 6: Depth of field effect; the camera is focused to the roof

Figure 7: Structural blur effect; the corners are more blurred than the center

6 Motion blur

In real world application, when a camera creates an image, the scene captured by the camera is not always static. Changes in the scene during the exposure are recorded to the image. Moving objects are blurred along their relative motion. This effect is called motion blur. Motion blur can be also caused by a camera motion. In such case, moving objects can be blurred and static objects must be blurred (the blur occurs along the change of the camera's viewport). An example of motion blur is shown in Figure 8. Motion blur is affected by the exposure time. Longer exposure time causes bigger blur effect.

The simulator expects to get input images representing of static scene with no motion blur even if the objects are in motion. The simulator uses fullscreen motion blur based on the algorithm in [13]. It allows us to simulate a camera motion.

Figure 8: Motion blur; the motion of the camera's view is simulated

We compute motion blur by a following equation per every pixel:

$$out = \frac{1}{n}\sum_{i=0}^{n-1} S\left(x_s + x_r\frac{i}{n-1}, y_s + y_r\frac{i}{n-1}\right), \quad (14)$$

$$x_r = x_d - x_s, \quad (15)$$

$$y_r = y_d - y_s, \quad (16)$$

where: out – output color pixel
n – number of steps
(x_s, y_s) – source coordinate of motion
(x_d, y_d) – destination coordinate of motion
S – input image

7 Sensor features

An output image is influenced by a sensor and its features. These features are noise, non-uniform response, and color filter array with demosaic filter. There are more features like blooming and artefacts caused by a transport in the sensor. In the current version, we simulate noise and color filter array with demosaic filter. Parameters of these features are known for many cameras unlike the other features.

7.1 Noise

Image noise is a random variation of brightness in images. The noise fundamentally limits the distinguishable content in the images. More information about CCD noise and evaluation of CCD sensor noise can be found in [9]. European machine vision association has also published EMVA Standard 1288[1] that describes the method of measurement and description noise of sold sensors and cameras. Simulated noise is shown in Figure 9.

In current version, we simulate noise the using:

$$B(x,y) = S(x,y) + r(f(S(x,y))), \quad (17)$$

Figure 9: Noise; temporal noise is added to the image

where: B – output image
S – input image
$r(i)$ – random number generator (e.g. Gaussian deistribution with a standard deviation i)
f – signal to noise ratio function

7.2 Color filter array

Most modern digital cameras acquire images using a single sensor overlaid with a color filter array. Each pixel on a camera sensor contains photo elements. The elemenents are monochromatic light sensitive and they do not distinguish wavelength of light. The output of the sensor is monochromatic image. Therefore, a color filter array is positioned on top of the sensor to filter out the component of light by the wavelength. The very common filter is the GRGB Bayer filter.

We simulate color filter array effect in two steps. We create a color mosaic image:

$$B_{color}(x,y) = S_{color}(x,y) \cdot M_{color}(x,y), \quad (18)$$

where: B – color mosaic image
S – input image
M – mosaic mask

The result of the color mosaicing is demosaiced by the following process:

$$B_{color} = S_{color} * K_{color}, \quad (19)$$

where: B – output image
S – input image
K – convolution kernel

If we want to simulate GRGB Bayer filter array, we use:

$$M_{red}(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ is odd} \\ 0 & \text{otherwise} \end{cases} \quad (20)$$

$$M_{green}(x,y) = \begin{cases} 1 & \text{if } x+y \text{ is odd} \\ 0 & \text{otherwise} \end{cases} \quad (21)$$

$$M_{blue}(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ is even} \\ 0 & \text{otherwise} \end{cases} \quad (22)$$

$$K_{blue} = K_{red} = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix} \quad (23)$$

$$K_{green} = \begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 1 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix} \quad (24)$$

In this case, simulated GRGB Bayer array is demosaiced by linear interpolation.

8 Conclusions and future work

This paper presents simulation of camera imperfections applied to computer generated images. The purpose of the simulation is to get computer generated images with features close to the features of images captured by real cameras. Such images can be used for image processing and computer vision applications testing. Some of the simulated imperfections can also be applied to high quality camera images to simulate output from lower quality cameras. The simulation of the camera imperfections is complex and presented solution still can be improved. Some of the features, such as distortion, chromatic abberation or vignetting can be simulated successfully. On the other hand, some others, such as lens flare, are very difficult to simulate because every camera has slightly different lenses that require individual and rather complex model.

Current version of the simulator is capable to process 0.9 frames per second. The test was performed with use of Intel(R) Core(TM)2 Duo P7350 2GHz on the image with resolution 640x480. In the future, we will optimize some of the algorithms, expecting an increase in performance.

Future versions of the simulator will allow simulation of more features, such as CCD sensor blooming, CCD streaking and more. Future works will also include modification of noise model. In order to improve the simulation, we will modify the implementation of the lens flare and we also will use more complex models of some imperfections.

9 Acknowledgements

I would like to thank Pavel Zemčík, Michal Košík, Peter Kubica, and Martin Krba for language corrections and valuable remarks. I also want to thank Oliver Zender from the Autrian Insitute of Technology for remarks and reviewing the article. This work was supported by the Artemis JU project R3-COP, grant no. 100233.

References

[1] *EMVA Standard 1288 - Standard for Characterization of Image Sensors and Cameras*. European Machine Vision Association, 3rd edition, 2010.

[2] John Owens Aaron Lefohn. Interactive depth of field using simulated diffusion. Technical report, 2006.

[3] Brian A. Barsky, Daniel R. Horn, Stanley A. Klein, Jeffrey A. Pang, and Meng Yu. Camera models and optical systems used in computer graphics: part i, object-based techniques.

[4] Brian A. Barsky and Todd J. Kosloff. Algorithms for rendering depth of field effects in computer graphics.

[5] Soon-Wook Chung, Byoung-Kwang Kim, and Woo-Jin Song. Detecting and eliminating chromatic aberration in digital images. In *Proceedings of the 16th IEEE international conference on Image processing*.

[6] Randima Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[7] Paul S. Heckbert. Fundamentals of Texture Mapping and Image Warping. Master's thesis, University of California, Berkeley, the United States of America, 1989.

[8] Evžen Hruška. *Fotografie na malý formát*. Státní nakladatelství technické literatury, 1959. L16-A-4-II/6227. (In Czech).

[9] Kenji Irie, Alan E. McKinnon, Keith Unsworth, and Ian M. Woodhead. A technique for evaluation of ccd video-camera noise. 2008.

[10] Todd J. Kosloff and Brian A. Barsky. Three techniques for rendering generalized depth of field effects. In *Proceedings of the Fourth SIAM Conference on Mathematics for Industry: Challenges and Frontiers (MI09)*.

[11] Xin Li, Bahadir Gunturk, and Lei Zhang. Image demosaicing: A systematic survey.

[12] Lili Ma, Yangquan Chen, and Kevin L. Moore. A new analytical radial distortion model for camera calibration. *CoRR*.

[13] Hubert Nguyen. *Gpu gems 3*. Addison-Wesley Professional, 1st edition, 2007.

[14] J. Žára, B. Beneš, J. Sochor, and P. Felkel. *Moderní počítačová grafika*. Computer Press, 2005. (in Czech).