A layered depth-of-field technique for handling partial occlusion in computer renderings

David C. Schedl

DIPLOMARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2011

© Copyright 2011 David C. Schedl

This work is published under the conditions of the *Creative Commons License Attribution–NonCommercial–NoDerivatives* (CC BY-NC-ND)—see http://creativecommons.org/licenses/by-nc-nd/3.0/.

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 21. September 2011

David C. Schedl

Contents

Acknowledgements

This thesis started as a project at the IT University Copenhagen, supervised by Paolo Purelli, and was continued as Master's thesis, supervised by Wilhelm Burger, at the University of Applied Sciences Upper Austria. With the hand-in of this work and the completion of my degree, a five-year long chapter of studying in Hagenberg, which I thoroughly enjoyed, closes. Thanks to everyone who supported me throughout those years.

Furthermore, I want to thank the following people for assistance with this thesis: Thanks to Christine Schedl-Kircher and Erwin Schedl for their support and patience and Melanie Hametner for her support and discussions. Thanks to Wilhelm Burger for supervision and providing helpful comments. Mariana Bernasconi helped me with proofreading this work. My friends and colleagues Thomas Berger, Christian Grossauer, Marko Jelen, Michal Karpowicz, Jürgen Koller, Sebastian Mayer, Tamara Nitsch, Beatrix Schwaiger, and Reto Stuber provided welcome diversions.

Abstract

Depth of field (DoF) represents a distance range around a focal plane in optic systems such as photographic cameras. Objects in camera-produced images out of this distance range appear to be blurred. The shape and amount of blurring depends on the configuration of the optical system. Depth of field is one of the effects which significantly contributes to the photorealism of images and therefore is simulated in rendered images. Photorealistic DoF can be achieved for non-real-time rendering applications with techniques such as ray-tracing.

Methods for rasterisation renderings try to approximate this realism at interactive rates. However, many rasterisation DoF implementations sacrifice quality for high frame rates and generate artefacts. One such artefact is produced by partial occlusion: blurry objects near the camera are semi-transparent and result in partially visible background objects. This effect is not achievable in single-layered renderings due to missing information in scene renderings. Partial occlusion can only be handled correctly if additional scene information is used. DoF methods, resolving this issue, render the scene into layers.

In this work DoF approaches are discussed and a method for solving partial occlusion is proposed. The contribution of this thesis is a layered method where the scene is rendered with a technique to retrieve hidden scene fragments. Rendered fragments are sorted into a set of layers which are blurred accordingly to their blurriness, determined by their circle of confusion, and composed by blending. The DoF effect is controlled by real world parameters known form photographic cameras such as focal length and f-stop. An approximation of DoF effects in optical systems without partial occlusion artefacts is thus produced.

Kurzfassung

Als Schärfentiefe wird ein Bereich bei optischen Systemen, wie Foto- und Filmkameras, bezeichnet. Objekte ausserhalb dieses Bereichs erscheinen unscharf auf Kamerabildern. Diese Unschärfe wird durch Parameter des optischen Systems definiert. Da Schärfentiefe in Foto und Film-kameras auftritt, ist sie ein wichtiger Bestandteil in fotorealistischen Bildern. Daher wird dieser Unschärfeeffekt oft bei computergenerierten Bildern simuliert. Methoden wie Raytracing erlauben fotorealistische Simulationen, sind aber zu rechenintensiv um diesen Effekt in Echtzeit zu berechnen.

Mit Rasterisierungsmethoden können interaktive Bildwiederholungsraten erzielt werden, jedoch können Schärfentiefeeffekte so nur approximiert werden, was Artefakte zur Folge hat. Ein Artefakt entsteht durch partielle Verdeckung: Objekte nahe der Kamera sind unscharf und werden dadurch transparent dargestellt; Hintergrundobjekte scheinen somit durch. Dieser Effekt ist mit Rasterisierungsmethoden nicht erzielbar, da verdeckte Bereiche nicht gerendert werden. Partielle Verdeckung kann nur richtig dargestellt werden, wenn zusätzliche Szeneninformationen bekannt sind. Methoden zur Simulation von Schärfentiefe lösen dieses Problem durch das Rendern von mehreren Schichten.

Diese Arbeit behandelt Schärfentiefemethoden in der Computer Grafik, welche das partielle Verdeckungsproblem lösen. Es wird eine eigene Lösung vorgestellt, in der die Szene mit einer Methode gerendert wird um verdeckte Pixel zu erhalten. Die gerenderten Pixel werden in Schichten sortiert und anhand ihrer Unschärfe gefiltert. Der Schärfentiefeeffekt wird durch Parameter wie Brennweite und Blendenzahl, welche aus der Fotografie bekannt sind, gesteuert. Dadurch wird eine gute Annäherung an Schärfentiefe inklusive partieller Verdeckung in optischen Systemen erzeugt.

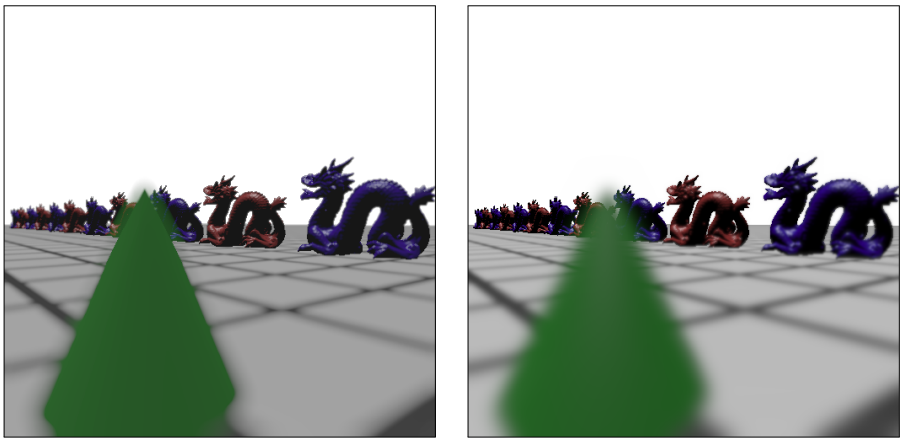
Chapter 1

Introduction

This thesis deals with the problem of simulating depth of field (DoF) for artificial images such as computer renderings. DoF represents a distance range around a focal plane in optic systems, such as camera lenses. Objects out of this range appear to be blurred compared to sharp objects in focus. This effect emphasises objects in focus and therefore is an important artistic tool in pictures and videos. The effect can be explained with a finite aperture camera model where all light-rays hitting the image plane travel through a optical lens. The lens has a particular focus point or focus range. Light-rays from objects out of this range are spread and therefore produce a smeared circle, also called the circle of confusion, instead of a sharp point on the image plane.

People in the field of computer graphics aim for the ambitious goal of generating photorealistic renderings. Depth of Field is one effect which significantly contributes to the photorealism of images because it is an effect that occurs in most optical systems. In computer renderings the pinhole-camera model, which rests upon the assumption that all light-rays travel through one point before hitting the image plane, is used. Therefore, there is no focus range and no smearing occurs resulting in a crisp image.

DoF can be simulated very accurately by ray tracing. However, ray tracers fail to produce accurate DoF effects at interactive frame rates. For interactive applications, the effect has to be simulated in real-time. Therefore, most approaches use fast post-processing techniques but sacrifice visual quality and produce artefacts in some situations. Post-processing methods work only on the rendered scene and usually do not need any further screen information apart from the depth buffer. Most common techniques to produce the DoF effect use an approach where pixels of the frame buffer get smeared according to their circle of confusion (CoC). The CoC is depending on the distance of objects and the lens parameters. One artefact in simple post-processing approaches is *partial occlusion*: An object in-focus behind an out-of-focus object should be partly visible at the blurred borders of the



(a) (b)

Figure 1.1: Depth-of-field effects produced with post-processing methods: a rendering produced with `Blender` (www.blender.org) showing artefacts (a); a rendering without partial occlusion artefacts, generated with the method described in chapter 5, (b). Note that the green cone, placed closely to the camera, smears to semi-transparency and reveals occluded dragons in (b), while in (a) the background is missing. Missing background information behind blurred objects is called *partial occlusion problem*, in DoF methods.

front object. Due to the lack of occluded pixels in rendered images, this effect cannot get simulated with simple post-processing approaches (shown in figure 1.1 (a)). In computer graphics, the used pinhole camera model is dismissing background pixels. Optical systems use a finite aperture camera model where light-rays from occluded objects can hit the image sensor.

This thesis presents an approach tackling the partial occlusion problem. By rendering the scene with a technique called depth peeling, hidden pixels can be retrieved. This hidden information is used to overcome the problem of partial occlusion (see figure 1.1 (b)). Rendered pixels are decomposed into layers. Thus, allowing each layer to be blurred uniformly and individually. Previous layered DoF methods produce discontinuity artefacts due to the layer splitting. In this thesis discontinuity artefacts are handled by smoothly decomposing layers. After blurring, the layers are composed by blending thus producing a rendering with appropriate simulated DoF and partial occlusion.

The rest of this thesis is structured as follows. In chapter 2 the generation of depth of field in optical systems is discussed. Chapter 3 gives an overview of rendering and texture mapping methods used in rasterisation rendering. Previous depth-of-field methods are discussed in chapter 4 and their artefacts, including partial occlusion, are explained. The approach proposed in this work is shown in chapter 5, with results and discussion shown in chapter 6. Conclusion and future work is presented in chapter 7.

Chapter 2

Optical systems and camera models

Computer renderings try to resemble the behaviour of optical systems such as cameras. Therefore this chapter deals with the optical constraints of cameras and camera parts. This thesis will not cover such devices in detail, but rather focus on important parts contributing to the depth of field i.e., lenses (see section 2.1) and the aperture stop (see section 2.2). In section 2.3 the depth of field and the impact of the lense and the aperture stop are discussed. Other important parts of cameras e.g., the shutter, the image sensor and the focusing unit are left out. For further details on cameras and optical systems one may refer to [30, 32].

The simplest setup for an optical camera is the pinhole camera model. This means that the camera is a box with a small hole. All rays pass through one point (the hole) and hit the image plane. The size of the hole is called aperture and is theoretically infinitely small. In practice, there are some limitations with small holes:

- Large apertures increase the amount of light but result in a blurry image due to large image spots.
- Small apertures darken the image and result in sharper images. However, diffraction[1] prevents the image spots from getting arbitrarily small.

These constraints make the pinhole model unusable for real cameras. Despite that, the pinhole model is a good and simple approximation of more complex cameras and therefore used in computer graphics (see section 3.1). The solution in modern optics for overcoming the limitations of pinhole cameras is to make the hole bigger and fill it with one or more lenses.

[1] The spreading of light after passing through a very small aperture, such as a pinhole or a narrow slit, is called *diffraction* [30, page 106].

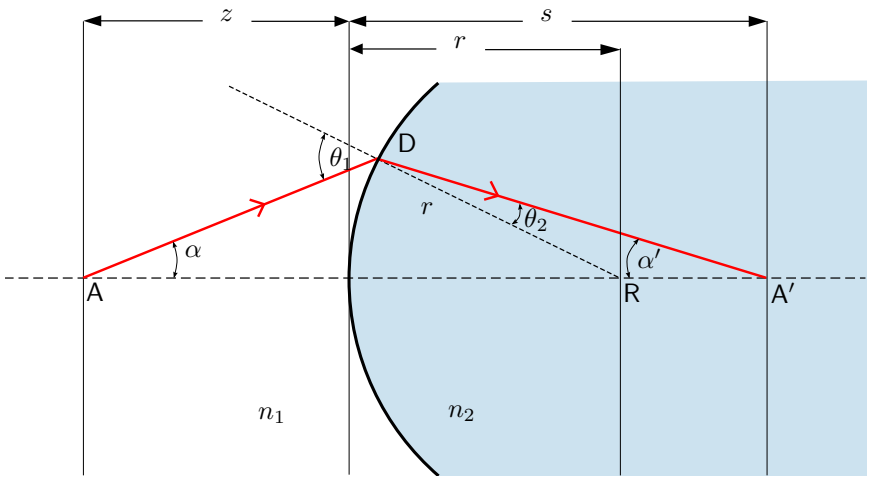


Figure 2.1: Refraction of a ray at a spherical surface (adapted from [30]).

2.1 Optical lenses

A lens is an optical device which transmits and refracts light. Lenses can be found in various systems such as microscopes, magnification glasses, video cameras, photo cameras, and even organic eyes. This section discusses models and theories for the use of lenses in cameras.

In cameras, the objective lens is used to transmit light-rays from the world onto an image plane where the light gets captured on a photographic film or digital sensor. In the following sections simple models of lenses will be discussed. Whereas modern camera lenses consist of more complex lens systems, simple lenses are sufficient for understanding the principles of depth of field. Further information on optical lenses can be found in [30] and [5].

2.1.1 Snell's law

The refraction of a light-ray within two optical mediums (e.g., lens glass and air) is defined by Snell's law.[2] Snell's law is defined as

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \frac{n_2}{n_1}, \tag{2.1}$$

where n_1, n_2 are the refraction indices[3] of two materials, θ_1 and θ_2 are the angles between the rays $\overline{\mathsf{AD}}$, $\overline{\mathsf{DA'}}$ and the surface normal (figure 2.1). Thus,

[2] *Snell's law* as defined in http://en.wikipedia.org/wiki/Snell's_law is a formula used in optics and physics to describe the relationship between the angles of incidence and refraction, when referring to light or other waves passing through a boundary between two different isotropic media, such as water and glass.

[3] The refractive index or index of refraction of a substance is a ratio of the speed of light in vacuum relative to that in the considered medium [30, pages 5–6].

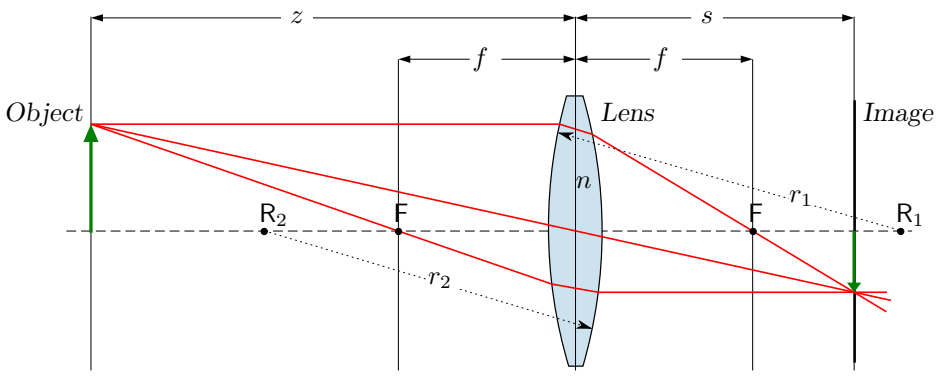


Figure 2.2: A thin-lens model (adapted from http://en.wikipedia.org/wiki/Lens_(optics)).

the following equation can be obtained

$$\frac{\sin(\alpha)}{\sin(\alpha')} = \frac{s-r}{z+r} \cdot \frac{n_2}{n_1}, \tag{2.2}$$

where z is the distance of the ray intersection with the optical axe in the material with refraction index n_1 (ray $\overline{\mathsf{AD}}$), s is the distance in material n_2 (ray $\overline{\mathsf{DA'}}$), and r is the radius of the curved surface (centre at point R) of the latter material. With equation 2.2 the image distances are not the same for all rays but are a function of the slope angle α. Thus, the rays do not come to a single focus. This is known as *spherical aberration* and is a well known problem of reflecting and refracting surfaces. If the angles are small enough, the so called *paraxial approximation* can be used. Paraxial approximation is where small sines are replaced by the angles themselves. If this is applied, equation 2.2 reduces to

$$\frac{n_1}{z} + \frac{n_2}{s} = \frac{n_2 - n_1}{r}. \tag{2.3}$$

2.1.2 Thin lens

A simple lens can be seen as an improved pinhole lens. With a thin lens the disadvantages of a pinhole model—i.e., either dark or blurred images—can be overcome. A thin lens focuses rays, from an object in the world, on the image plane allowing for sharp images with large apertures, and resulting in brighter images. The curvature of a lens is defined by its radii. Figure 2.2 shows a biconvex model of a thin lens with the radii r_1 and r_2 whose centre points are R_1 and R_2 respectively. Considering a single thin lens in a homogeneous medium, it can be shown that the object and image distances

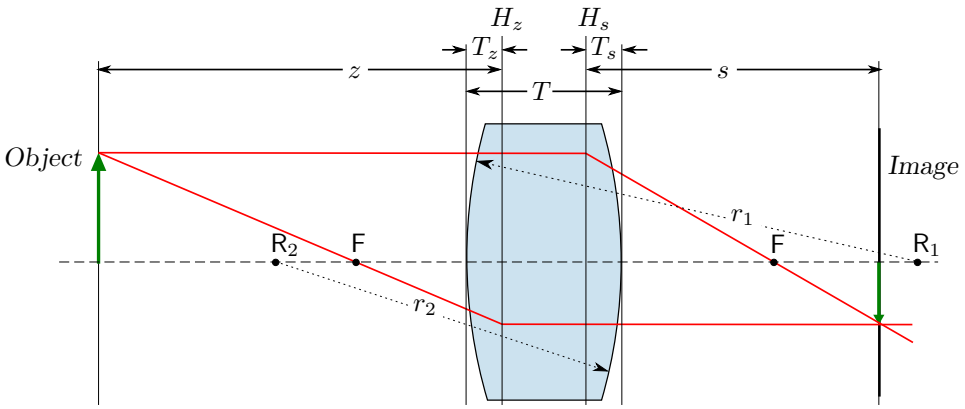


Figure 2.3: A thick lens with the focal point F. The distances are measured from the *principal planes* H_z and H_s (inspired by [30]).

for paraxial rays are related by the equation

$$\frac{1}{z} + \frac{1}{s} = (n-1)\left(\frac{1}{r_1} - \frac{1}{r_2}\right), \tag{2.4}$$

where n is the index of refraction for the lens' material, z and s are the distances in front and behind the lens respectively. The focal length f is defined as the image distance for parallel incoming rays. Thus, light-rays parallel to the optical axis pass the point F. For a thin lens f is defined by

$$\frac{1}{f} = (n-1)\left(\frac{1}{r_1} - \frac{1}{r_2}\right). \tag{2.5}$$

This equation is known as the *lens-maker's formula* [30]. Note that the distances z and s (equation 2.4) are fixed lengths. Moreover, only objects at distance z produce sharp projections at the image plane at distance s.

2.1.3 Thick lens

The model of a thin lens implies that the medium is infinitely thin. This is not applicable for real lenses such as the one shown in figure 2.3. Therefore, the distances (i.e., s, z and s) are measured from the *principal planes* H_z and H_s at which the refraction can be considered to happen. The focal length is calculated from the equation

$$\frac{1}{f} = (n-1)\left(\frac{1}{r_1} + \frac{1}{r_2} - \frac{(n-1)^2 \cdot T}{n \cdot r_1 \cdot r_2}\right), \tag{2.6}$$

where T is the lens thickness. The positions of the principal planes are given by

$$T_z = f \cdot T \cdot \left(\frac{1-n}{r_2}\right) \qquad \text{and} \qquad T_s = f \cdot T \cdot \left(\frac{1-n}{r_1}\right). \tag{2.7}$$

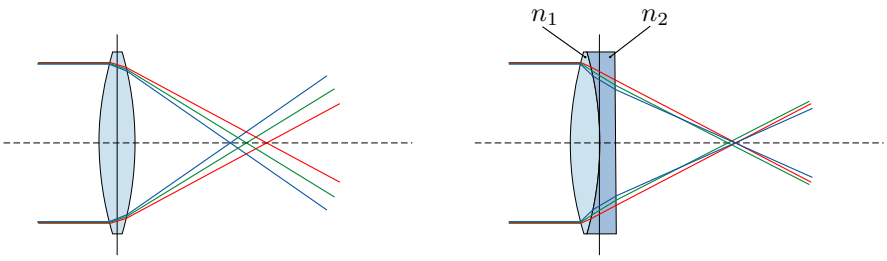


Figure 2.4: A thin lens and the effects of *chromatic aberration*. Due to dependence of the refraction index on the wavelength of the light, different colours get refracted differently (a). A second lens with a different dispersion, right after the first lens, can reduce this artefact (b) (from http://en.wikipedia.org/wiki/Lens_(optics)).

2.1.4 Compound lens

In camera objectives, there is more than one kind of lens used to prevent the effects of chromatic aberration. Furthermore, the focal distance can be varied to focus on a certain distance (in the coming sections referred to as z_{focus}). If thin lenses get combined, the resulting focal distance f can be expressed as

$$\frac{1}{f} = \frac{1}{f_1} + \frac{1}{f_2} + \frac{1}{f_3} + \dots, \tag{2.8}$$

where $f_1, f_2, \dots$ are the focal lengths of the individual lenses placed closed together (in contact). If the lenses are separated by a distance E, the effective focal length of two lenses with the focal lengths f_1 and f_2 are related by

$$\frac{1}{f} = \frac{1}{f_1} + \frac{1}{f_2} - \frac{E}{f_1 \cdot f_2}. \tag{2.9}$$

Chromatic aberration

The focal length of a simple lens (equation 2.5) varies with the wavelength of the light. The index of refraction n is not uniform for all wavelengths of the visible light. This variation is called chromatic aberration and can be reduced by combining lenses with different refractive indices. Figure 2.4 (a) shows the effect and figure 2.4 (b) shows how to substantially avoid it.

2.2 Aperture stop

With lenses, it is possible to bundle more light on an image sensor thus producing brighter images. However, in photographic cameras it is desirable

to control the amount of light passing the lens. A diaphragm placed in front of or behind a lens, in combination with the shutter, is used to regulate the exposure. The aperture also cuts off oblique light rays.

Photographic lenses specify their diaphragm settings as f-stops determined by dividing the lens' focal length by the diameter of the aperture. Its relationship to exposure regulation is directly proportional to its area. Since the doubling or halving of the exposure is related to the multiplication or division of the diameter by the square root of two, this serves as the basis for the series of f-stops such as the values $1.0, 1.4, 2.0, 2.8$, etc. The diameter of the aperture a is defined as

$$a = \frac{f}{N} \tag{2.10}$$

where N is the f-stop value as described above. Ideally, the shape of the diaphragm is a circle. However limited by mechanical constraints, the aperture often has a different shape, e.g., a rectangular, hexagonal or octagonal shape. The form of the diaphragm is responsible for the shape of the out-of-focus blur, and is most observable at bright out-of-focus spots, e.g., at lights and reflections. This effect is known as *bokeh effect* and the function defining the shape of the blur is often called a *point spread function* (PSF) in computer graphics.

Further details and also some historical background on diaphragms can be found in [32]. The aperture also has an important impact on the DoF effect which will be discussed in section 2.3.

2.3 Depth of field

Hypothetically, a sharp image point will only appear on the image plane from an object exactly in focus, located at z_{focus} (see figure 2.5). In practice, because of the limitations of the human eye, objects within an acceptable sharpness are recognized as sharp. Thus, those points on the image plane are small enough and appear to be in foucs. DoF is defined as the distance range between the closest and furthest objects with acceptable sharpness on the image plane [24]. If a lens is focused on an object in front of the camera at distance z_{focus}, the object lies within this sharpness range. In figure 2.5, the in-focus area is the area between the points A and B. The diameter of the sharpness area is symbolized as $\overline{\mathsf{MN}}$ and will be referred to as d_z. Line $\overline{\mathsf{DE}}$ is the diameter of the aperture a. If d_z is located at the distance z_{focus} from the lens, then the range within acceptable sharpness lies within z_{front} and z_{back}. Since $\triangle\mathsf{ADE}$ is similar to $\triangle\mathsf{AMN}$ and $\triangle\mathsf{BDE}$ is similar to $\triangle\mathsf{BMN}$ the following equations can be obtained [62]:

$$\frac{z_{\text{focus}} - z_{\text{front}}}{d_z} = \frac{z_{\text{front}}}{a}, \qquad \frac{z_{\text{back}} - z_{\text{focus}}}{d_z} = \frac{z_{\text{back}}}{a}. \tag{2.11}$$

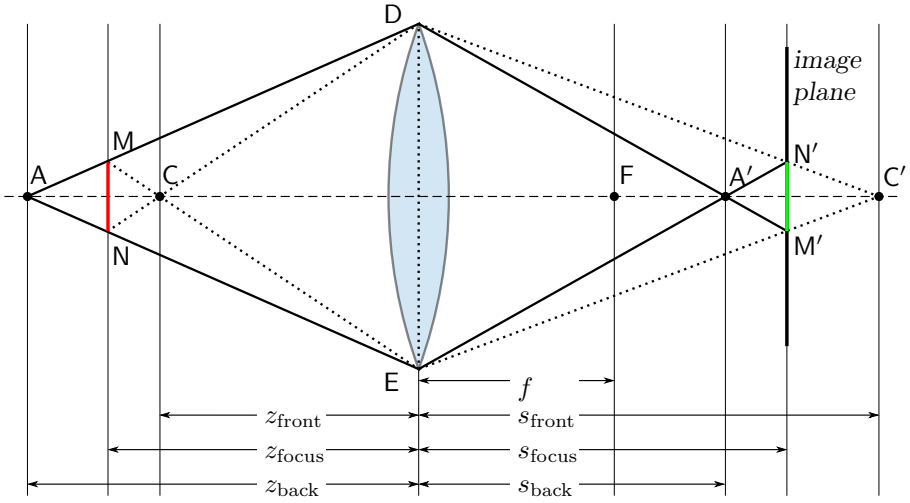


Figure 2.5: The range of Depth of Field in front and behind the lens. There are similar triangles around the diameters $\overline{\mathsf{MN}}$ and $\overline{\mathsf{N'M'}}$, which can be used to calculate these distances (inspired by [62]).

Although z_{front} and z_{back} can be determined if d_z is given, it is unusual to work with the diameter on the object side of the lens. The diameter on the image plane d_{coc} is called *circle of confusion* and better suited for the purpose of this work. In figure 2.5 the length of $\overline{\mathsf{N'M'}}$ defines this circle of confusion. The triangle $\triangle\mathsf{C'N'M'}$ is similar to $\triangle\mathsf{C'DE}$ and $\triangle\mathsf{A'N'M'}$ is similar to $\triangle\mathsf{A'DE}$, resulting in

$$\frac{s_{\text{focus}} - s_{\text{back}}}{d_{\text{coc}}} = \frac{s_{\text{back}}}{a} \qquad \text{and} \qquad \frac{s_{\text{front}} - s_{\text{focus}}}{d_{\text{coc}}} = \frac{s_{\text{front}}}{a}. \tag{2.12}$$

Now the circle of confusion can be obtained as

$$d_{\text{coc}}(s, a, s_{\text{focus}}) = |s_{\text{focus}} - s| \cdot \frac{a}{s}, \tag{2.13}$$

where s can be any distance behind the lens. For a photographic lens, measuring the distance z instead of s is common because z specifies the length in front of the lens to an object. From equations 2.4 and 2.5 we get

$$\frac{1}{z} + \frac{1}{s} = \frac{1}{f}, \tag{2.14}$$

and, thus

$$s = \frac{f \cdot z}{z - f}. \tag{2.15}$$

Equation 2.13 can now be rearranged to

$$d_{\text{coc}}(z, f, N, z_{\text{focus}}) = \left| \frac{f^2 \ (z - z_{\text{focus}})}{z\, N \ (z_{\text{focus}} - f)} \right|, \tag{2.16}$$

where z is the distance to the object in front of the lens and N is the f-stop number.

The observation that the aperture has an important impact on the circle of confusion can be made now. A small aperture, thus a high N, increases the depth of field resulting in a small CoC diameter as shown in figure 2.6 (b). In figure 2.6 (a) an open diaphragm is shown resulting in a higher blurring of out of focus objects. The ideal projection (i.e., no blurring) with a pinhole model is shown in figure 2.6 (c). Figures 2.7 (a)–(c) show some plots of equation 2.16 and the influence of different parameters on the circle of confusion. Figure 2.7 (a) shows the influence of changing the focal length f, figure 2.7 (b) the impact of modifying N, and figure 2.7 (c) shows the difference of the CoC-plot if z_{focus} is changing.

Hyperfocal distance

One distance setting, called *hyperfocal distance*, is quite popular when talking about DoF. It is used in DoF techniques such as [24, 62, 68] and defined in Goldberg's textbook [32] as

> The hyperfocal distance is the distance setting at a given f-number that produces a depth-of-field with infinity at the far limit and half the hyperfocal distance at the near limit. It's the focus setting on nonadjustable snapshot cameras.

The *hyperfocal distance* B is approximated by

$$B = \frac{f^2}{N \cdot d_B}, \tag{2.17}$$

where d_B is the accepted diameter; the diameter at which image points still appear to be in focus. In [32] a sharpness standard of $d_B = 0.03$ mm is used.

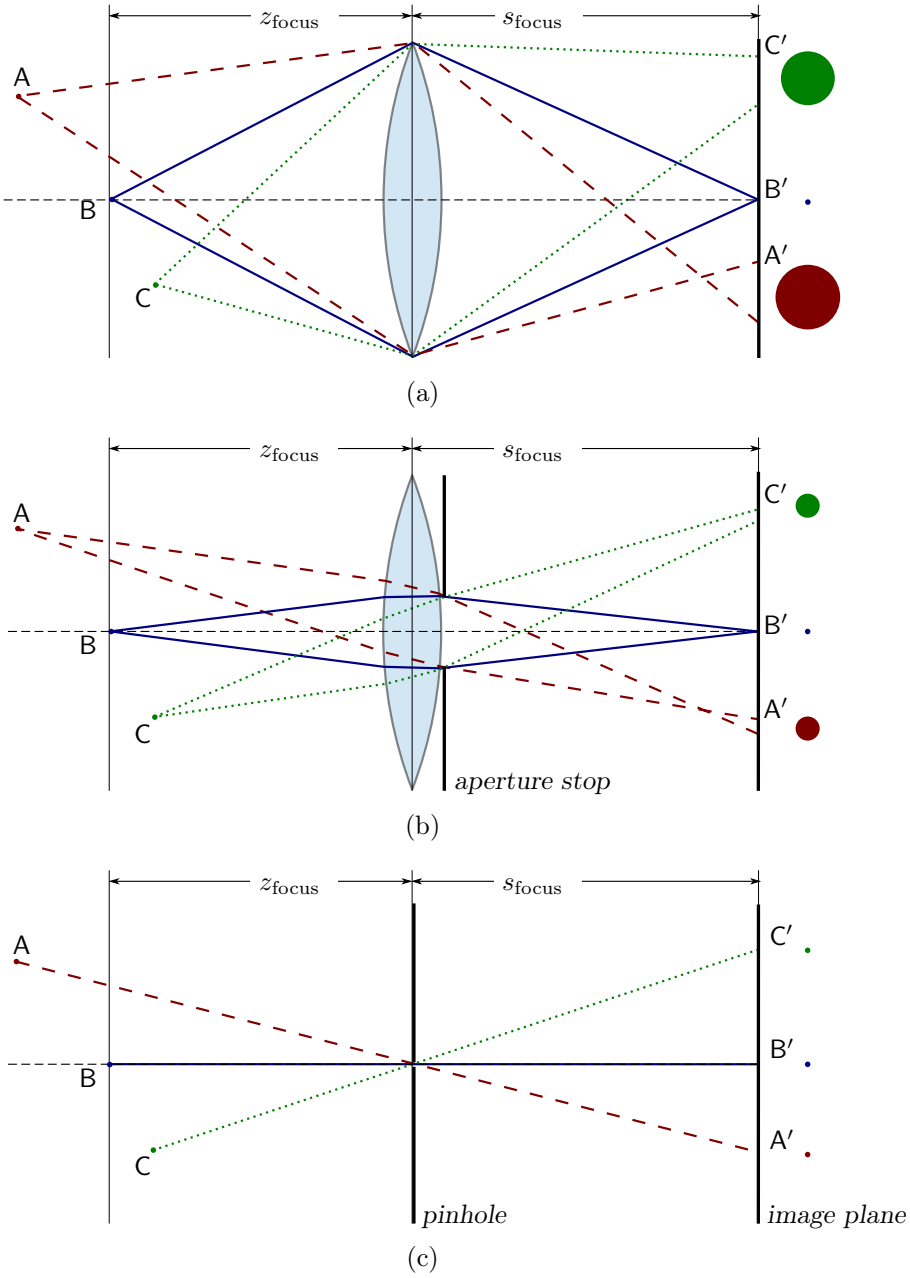


Figure 2.6: A lens with an open aperture and 3 points $\mathsf{A}, \mathsf{B}, \mathsf{C}$ and their projections $\mathsf{A}', \mathsf{B}', \mathsf{C}'$ on the image plane (a). The same with a closed aperture (b). For comparison the ideal projections (small spots) with a pinhole camera model are shown (c) (adapted from [32] and http://en.wikipedia.org/wiki/Depth_of_field).

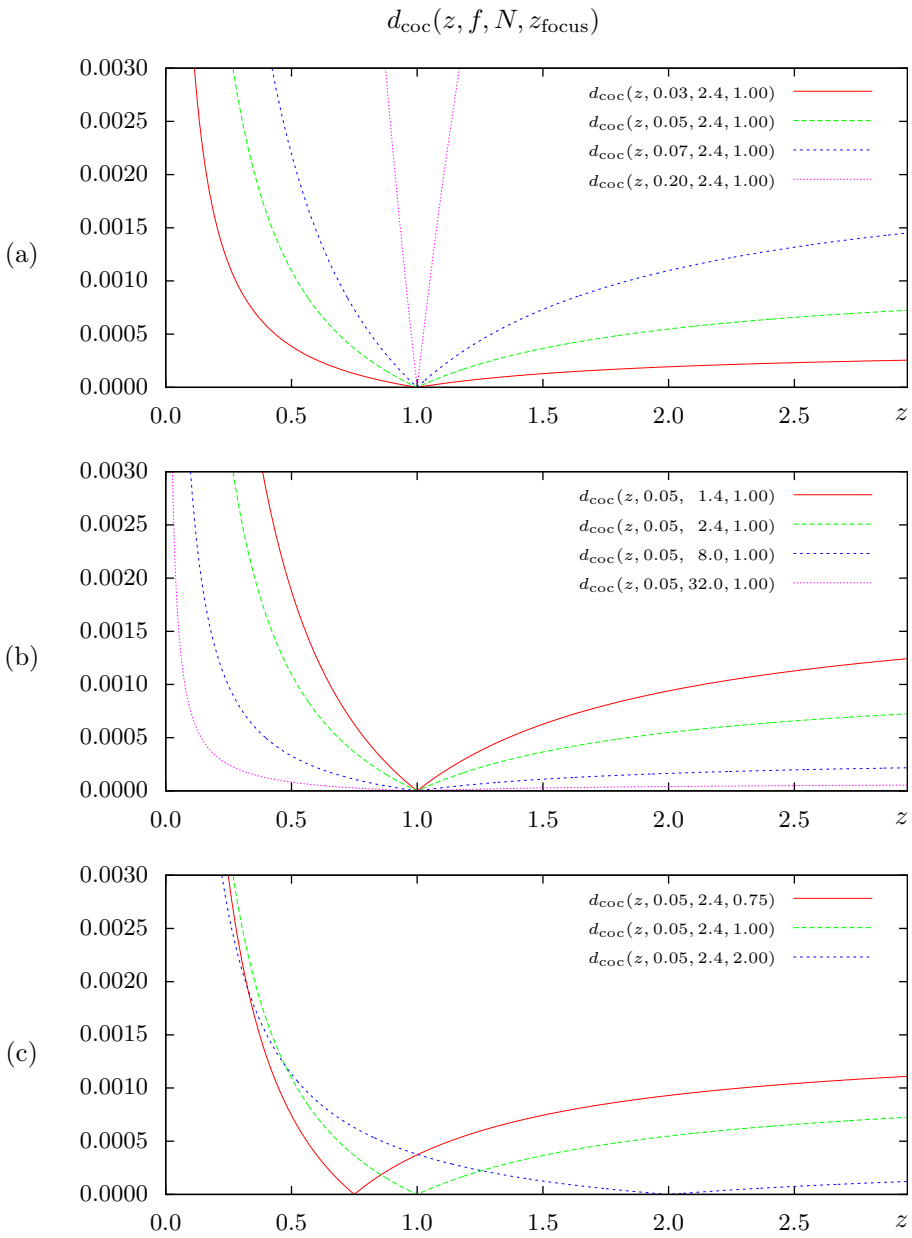


Figure 2.7: A plot of $d_{\text{coc}}(z, f, N, z_{\text{focus}})$, equation 2.16, and the impact of changing parameters: the focal length f (a); the f-stop number denoted by N (b); the focal plane at z_{focus}. The units of the axis are shown in meters.

Chapter 3

Rendering basics and post processing

In chapter 2, the optical constraints of DoF are discussed. Since this thesis deals with depth-of-field simulations for computer generated images, some principles of computer graphics such as rendering, post-processing, and filtering are discussed in this chapter. Further readings regarding these topics can be found in [2, 11, 25, 26, 29, 57, 59, 64].

In rendering applications, a scene is composed of points in 3D (vertices), edges between them, and polygonal faces. Since dealing with homogeneous coordinates[1], transformations are often matrix-multiplications with 4×4 dimensional matrices. Vertices are rasterised for displaying a scene on the screen. These operations can be done efficiently on modern GPUs which are highly optimized for such purposes. One popular application for these 4×4 matrices is the transformation of points into different coordinate systems. A point in a 3D scene is often defined in a global coordinate system. With matrix transformations, it can be transformed into the view space where the camera position is at the origin. From view space, the point can be easily projected onto a image plane. The next sections deal with camera projections (section 3.1), post processing (section 3.2), blending (section 3.3), and filtering (section 3.4). Again, it is not the scope of this thesis to discuss the rendering pipeline in detail but rather to give a rough overview with respect to this work.

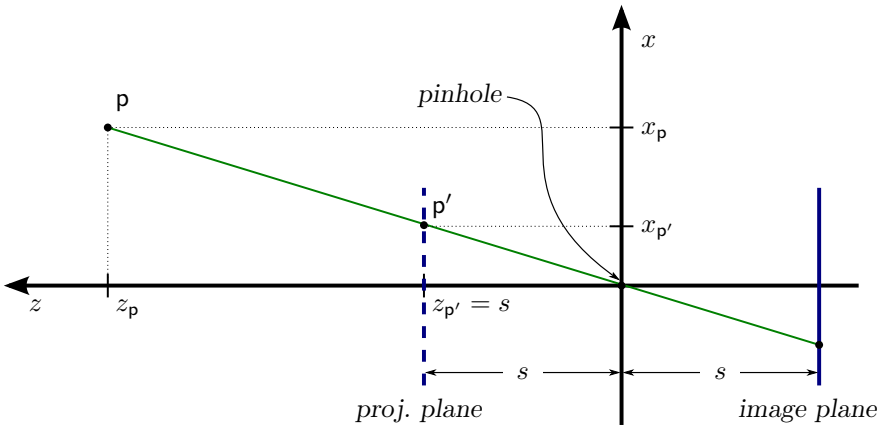


Figure 3.1: The notation for deriving a simple perspective projection matrix. A perspective projection of point p onto the *projection plane* at $z = s$ yields to the projected point p′. The projection is performed from the origin, where the pinhole of a pinhole camera would be located. The projection of point p onto an image plane of a pinhole camera would lead to a projection on the *image plane* which is a mirrored version of p′. Both planes are located a distance s away from the origin.

3.1 Camera projection

In chapter 2 the parts of an optical camera, their impact on depth of field, and the concept of a pinhole camera are discussed. Although the pinhole model is not used in real cameras, this simple concept is used in computer graphics. The model of all rays passing one point, the pinhole, and hitting the image plane always in perfect focus can be calculated with a projection. Since it is possible to perform projections with 4×4 matrices, this model seems perfect. A perspective projection is used more closely because it resembles how the human eye perceives the world, i.e., objects further away are smaller. Projecting a point $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p}, z_\mathsf{p})^T$ onto a plane at $z = s$, $s > 0$ yields to a new point $\mathsf{p}' = (x_{\mathsf{p}'}, y_{\mathsf{p}'}, s)^T$ as shown in figure 3.1. With similar triangles, the following equations for the components of p can be obtained

$$x_{\mathsf{p}'} = s\frac{x_\mathsf{p}}{z_\mathsf{p}} \qquad \text{and} \qquad y_{\mathsf{p}'} = s\frac{y_\mathsf{p}}{z_\mathsf{p}}. \tag{3.1}$$

[1] *Homogeneous coordinates* are a system of coordinates used in projective geometry. They have the advantage that the coordinates of points, including points at infinity, can be represented using finite coordinates. Expressions involving homogeneous coordinates are often simpler (from http://en.wikipedia.org/wiki/Homogeneous_coordinates).

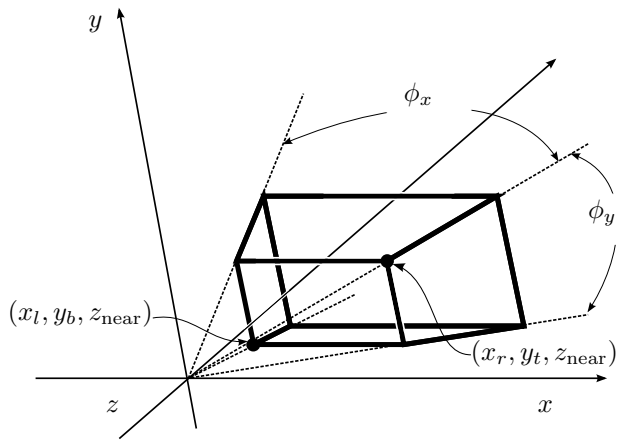


Figure 3.2: The view frustum in view space (from [2]).

With the use of homogeneous coordinates this calculation can be packed into the 4×4 matrix [2]

$$\mathbf{M}_{\text{proj}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/s & 0 \end{pmatrix}. \tag{3.2}$$

A point p can be transformed into a projected space (as shown in equation 3.1), with matrix $\mathbf{M}_{\text{proj}}$ from equation 3.2, as

$$\mathsf{p} = \mathbf{M}_{\text{proj}} \cdot \mathsf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/s & 0 \end{pmatrix} \cdot \begin{pmatrix} x_{\mathsf{p}} \\ y_{\mathsf{p}} \\ z_{\mathsf{p}} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{\mathsf{p}} \\ y_{\mathsf{p}} \\ z_{\mathsf{p}} \\ z_{\mathsf{p}}/s \end{pmatrix} \equiv \begin{pmatrix} s\, x_{\mathsf{p}}/z_{\mathsf{p}} \\ s\, y_{\mathsf{p}}/z_{\mathsf{p}} \\ s \\ 1 \end{pmatrix}.$$

The perspective projection in equation 3.2 assumes that the eye-point is at the origin and all points are projected on a plane. However, rather than projecting onto a plane it is more desirable to transform the view frustum into a view volume. A view frustum can be defined by the parameters $(x_l, x_r, y_b, y_t, z_{\text{near}}, z_{\text{far}})$ defining planes in the world space such as shown in figure 3.2 where $0 < z_{\text{near}} < z_{\text{far}}$. The horizontal field of view ϕ_x is determined by the angle between the left and the right planes. The vertical field of view ϕ_y is defined by the angle between the upper and lower plane. These planes are determined by x_l, x_r and y_t, y_b, respectively. The perspective ma-

trix transforming the view frustum into the view volume is [2]

$$\mathbf{M}_{\text{volume}} = \begin{pmatrix} \frac{2z_{\text{near}}}{x_r - x_l} & 0 & -\frac{x_r + x_l}{x_r - x_l} & 0 \\ 0 & \frac{2z_{\text{near}}}{y_t - y_b} & -\frac{y_t + y_b}{y_t - y_b} & 0 \\ 0 & 0 & \frac{z_{\text{far}} + z_{\text{near}}}{z_{\text{far}} - z_{\text{near}}} & -\frac{2z_{\text{far}} z_{\text{near}}}{z_{\text{far}} - z_{\text{near}}} \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.3}$$

Instead of simply setting $z_{\mathsf{p}'} = s$ (as in equation 3.2), by projecting a point with the matrix $\mathbf{M}_{\text{volume}}$ the depth of a projected point is preserved. Depth values z in the view frustum are mapped to the range $[-1, 1]$ determined by the near- and far-clipping planes z_{near} and z_{far}. Such planes at the front and back of the view frustum define a finite depth range $z_{\text{near}} < z_{\mathsf{p}} < z_{\text{far}}$ for a point p. Objects in the scene in front or behind the frustum are clipped. The mapping and clipping of fragments has the advantage of better using the precision of floating-point numbers; thus preventing artefacts such as z-fighting.[2] However, one characteristic of the perspective transform matrix is the greater precision for objects closer to the viewer [11, chapter 18].

Note that there are variations of the view volume projection matrix (equation 3.3) depending on the intervals of the volume used; e.g., the depth interval typically is $[-1, 1]$ in OpenGL but $[0, 1]$ in DirectX.[3] Additionally, the coordinate systems can be right-handed or left-handed. In right-handed coordinate systems the z-axis points towards the viewer. This means that an infinitely far away point p, from the viewer's location, is located at $z_{\mathsf{p}} = -\infty$ while in a left-handed coordinate system the point is located at $z_{\mathsf{p}} = \infty$. Since the latter system is more natural when working optical-camera coordinates, figures in this work use left-handed coordinate systems. Matrices only need small modifications to transform between left-handed and right-handed coordinate systems.

Field of view angle

A natural way for specifying the view frustum is by the field of view angles ϕ_x and ϕ_y as shown in figure 3.2. The formula for converting from camera lens size to field of view is defined as

$$\phi_x = 2 \arctan\left(\frac{w}{2f}\right), \tag{3.4}$$

[2] In [2, page 883], *z-fighting* is explained as: "For example, say you modelled a sheet of paper and placed it on a desk, ever so slightly above the desk's surface. With precisions limits of the z-depths computed for the desk and paper, the desk can poke through the paper at various spots. This problem is sometimes called *z-fighting*."

[3] OpenGL (Open Graphics Library) and DirectX are standard specifications for applications producing 2D or 3D computer graphics, developed by the Khronos group and the Microsoft Corporation, respectively (from http://www.opengl.org and http://en.wikipedia.org/wiki/DirectX.

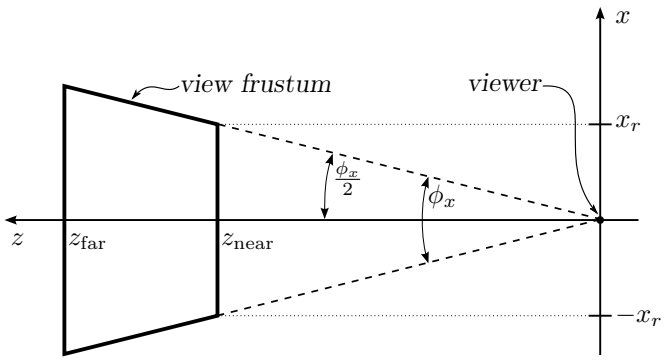


Figure 3.3: Top view of a symetrical view frustum.

where ϕ_x is the horizontal field of view, w is the horizontal frame/sensor size, and f the focal length. In rendering applications w is often a constant; e.g., $w_{\text{Blender}} = 32$ mm in Blender.[4] If the view frustum is symmetric ($x_r = -x_l$ and $y_t = -y_b$), as shown in figure 3.3, the following can be derived:

$$\tan\left(\frac{\phi_x}{2}\right) = \frac{x_r}{z_{\text{near}}}. \tag{3.5}$$

Thus the projection matrix (equation 3.3) can be simplified to

$$\mathbf{M}_{\text{fov}} = \begin{pmatrix} \frac{1}{\tan(\phi_x/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\phi_y/2)} & 0 & 0 \\ 0 & 0 & \frac{z_{\text{far}}+z_{\text{near}}}{z_{\text{far}}-z_{\text{near}}} & -\frac{2z_{\text{far}}z_{\text{near}}}{z_{\text{far}}-z_{\text{near}}} \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.6}$$

In pbrt the perspective projection matrix is calculated this way [60, page 312].

3.2 Post processing

Renderings are often produced by projecting polygons, as discussed in the previous section (section 3.1) and rasterising these projections. If a rendering is based on an image rather than on polygons it is referred to as *image-based rendering*. The advantage of such representations is that the rendering costs are proportional to the number of pixels and not dependant on the complexity of a scene; e.g., defined by the number of polygons.

In [2] the goal of rendering is defined as follows:

> The goal of rendering is to portray an object on the screen; how we attain that goal is our choice. There is no single correct way

[4] Blender is an open source 3D modelling application (www.blender.org).

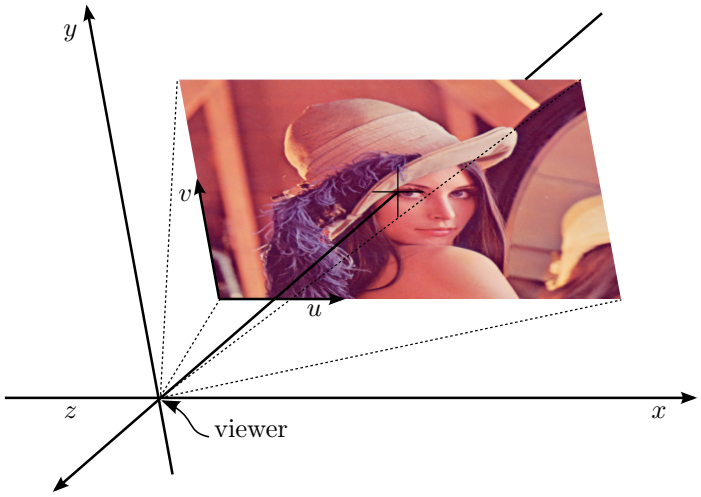


Figure 3.4: The view space (x, y, z), where the viewer's position is at the origin. The dotted lines represent the frustum. A screen spaced aligned quad with the texture space (u, v) is shown.

to render a scene. Each rendering method is an approximation of reality, at least if photorealism is the goal.

The spectrum of image based renderings ranges from simple appearance based methods such as sprites renderings, and layer compositions to more complex, physically based ones such as *global illumination*.[5] In [2] the author provides a good overview of techniques.

If an image based method is applied to a rendered scene before showing the result on the screen, the method is referred as *post processing* method. Such methods might also be called *screen spaced* methods; this is in reference to the fact that operations are done on objects in the projected space.

Screen spaced techniques

Screen spaced techniques use the renderings of a scene which are then in some form rendered to an off-screen buffer,[6] containing colour, depth, normals and so on. When post processing is used on GPUs, these buffers are treated as textures and applied to screen-filling quadrilaterals (as shown in

[5] *Global illumination* is a group of algorithms used in 3D computer graphics to add more realistic lighting to 3D scenes. Such algorithms take into account not only the light which comes directly from a light source (direct illumination), but also light rays reflected by other surfaces in the scene—indirect illumination (http://en.wikipedia.org/wiki/Global_illumination).

[6] A *off-screen buffer* is a buffer which's content is initially not intended to be shown on a screen.

figure 3.4). Post processing is performed by rendering the quad and applying programmable pixel shaders on the quad. Various effects can then be applied.

Special care has to be taken when converting between the texture space (u, v) and the viewport coordinates [70]. The x and y coordinates on the projection plane are in the range $[-1, 1]$ while the u and v coordinates of the screen spaced quadrilaterals are defined on the interval $[0, 1]$. The off-screen buffers are usually of the same dimension as the screen. Depending on the hardware, buffers can use 8-bit, 16-bit or even 32-bit per channel and also vary in the number of channels. The z-depth buffer is usually stored with 16-bit or 32-bit precision. The four channel colour information (red, green, blue, and alpha) of a rendered scene is often stored with 8-bit per channel. In modern render pipelines, the colour information is saved with higher precision for use in HDR images. Further details on image processing and post processing on the GPU can be found in [2, 10, 54].

Deferred rendering takes screen spaced approaches one step further and applies operations which are often done during rasterisation, such as lighting and normal mapping, as a post process [71].

3.3 Alpha blending

Rendering algorithms, either polygon (section 3.1) or image based (section 3.2), write their results in the frame buffer. If there is already a colour value in the frame buffer, the previous values have to be combined with the new one. This process is often referred as blending. Blinn in [12, chapter 16] describes it as follows:

> One of the most important antialiening [!] tools in computer graphics comes from a generalization of the simple act of storing a pixel into the frame buffer. Several people simultaneously discovered the usefulness of this generalization so it goes by several names: matting, image compositing, alpha blending, overlaying, or lerping.

One of the first publications covering this problem is [61] by Porter and Duff. The authors extend the representation of a three-valued RGB-pixel by a fourth component named a "coverage" value α.

3.3.1 Alpha channel

With the additional α component per pixel, the representation of a pixel $\mathbf{p}$ is defined as

$$\mathbf{p} = (r, g, b, \alpha)\,, \tag{3.7}$$

where r, g, b are the colour values and α is a coverage component. In this section, the range of the colour components is defined in the range $[0, 1]$

where 1 in the alpha channel indicates coverage and 0 indicates no coverage. Simply said, with this representation it is possible to model transparency. Some examples are

$$\begin{aligned}\textbf{red} &= (1,0,0,1),\\ \textbf{white} &= (1,1,1,1),\\ \textbf{black} &= (0,0,0,1),\\ \textbf{clear} &= (0,0,0,0).\end{aligned}$$

Of special interest are **black** and **clear** where the former is a opaque black and the latter is a transparent pixel. One obvious way to deal with semi-transparent colours would be to modify the alpha component, such as $(1,0,0,0.5)$ for a semi-transparent red. However, for some applications this representation is not the best way (further discussed in section 3.3.3).

Porter and Duff discuss various blending operators using the α component. One operator frequently used for blending is the *over* operator.

3.3.2 Over operator ($\oplus$)

To overlay a foreground image on top of some background image the *over* operator can be used. Say the background colour is defined as $\mathbf{B}$ and the foreground as $\mathbf{F}$, where both colours are a three-element vectors. With the over operator the composited colour value $\mathbf{B}'$ is defined as

$$\mathbf{B}' = (1-\alpha)\mathbf{B} + \alpha\mathbf{F}, \tag{3.8}$$

where α is the coverage component of the foreground [61].

However, this is not general enough when dealing with two semi-transparent colour values $\mathbf{F}$ and $\mathbf{G}$ with the occlusion values α and β, respectively. A pixel colour $\mathbf{G} \oplus \mathbf{F}$—"$\mathbf{G}$ over $\mathbf{F}$"—is defined as

$$\mathbf{G} \oplus \mathbf{F} = \frac{\alpha(1-\beta)\mathbf{F} + \beta\mathbf{G}}{\gamma}, \tag{3.9}$$

where

$$\gamma = \alpha + \beta - \alpha\beta \tag{3.10}$$

is the alpha value of $\mathbf{G} \oplus \mathbf{F}$ [12, chapter 16]. The result of the operation is shown in figure 3.5 (c). Figure 3.5 (d) shows the result of $\mathbf{F} \oplus \mathbf{G}$. Note that the blending order is important.

If $\mathbf{F}$ is an opaque colour ($\alpha = 1$) the equation unfolds to equation 3.8

$$\mathbf{G} \oplus \mathbf{F} = (1-\beta)\mathbf{F} + \beta\mathbf{G}. \tag{3.11}$$

Since a colour value is often multiplied by its alpha component, such as $\alpha\mathbf{F}$ and $\beta\mathbf{G}$, one key insight of [61] is the association of the colours with their opacity.

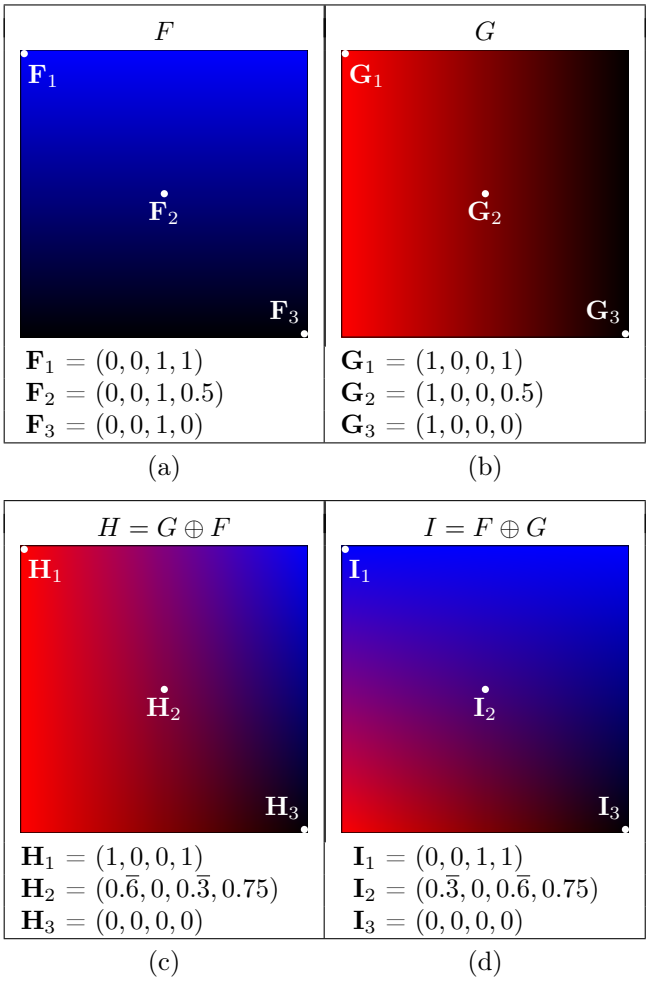


Figure 3.5: Two images F (a), G (b) and the composition of those two with the *over operator* (c), (d). For each image three sample colour values, in the form of a quadruple (equation 3.7) are shown.

3.3.3 Premultiplied alpha

The simplification of multiplying a pixels colour by their opacity is usually referred to as opacity *associated* with the colour, or as having the colour *premultiplied* by its alpha value (mentioned in [61] and cited in [12, chapter 16]). *Premultiplied* colour values are defined as:

$$\tilde{\mathbf{F}} = \alpha\mathbf{F}, \tag{3.12}$$

$$\tilde{\mathbf{G}} = \beta\mathbf{G}, \tag{3.13}$$

$$\tilde{\mathbf{G}} \oplus \tilde{\mathbf{F}} = \gamma(\mathbf{G} \oplus \mathbf{F}). \tag{3.14}$$

$\tilde{\mathbf{F}}_1 = (0, 0, 1, 1)$	$\tilde{\mathbf{G}}_1 = (1, 0, 0, 1)$
$\tilde{\mathbf{F}}_2 = (0, 0, 0.5, 0.5)$	$\tilde{\mathbf{G}}_2 = (0.5, 0, 0, 0.5)$
$\tilde{\mathbf{F}}_3 = (0, 0, 0, 0)$	$\tilde{\mathbf{G}}_3 = (0, 0, 0, 0)$
(a)	(b)
$\tilde{\mathbf{H}}_1 = (1, 0, 0, 1)$	$\tilde{\mathbf{I}}_1 = (0, 0, 1, 1)$
$\tilde{\mathbf{H}}_2 = (0.5, 0, 0.25, 0.75)$	$\tilde{\mathbf{I}}_2 = (0.25, 0, 0.5, 0.75)$
$\tilde{\mathbf{H}}_3 = (0, 0, 0, 0)$	$\tilde{\mathbf{I}}_3 = (0, 0, 0, 0)$
(c)	(d)

Table 3.1: The two images F and G, (a) and (b) and their compositions H and I, (c) and (d) from figure 3.5, converted to premultiplied colour values (see equation 3.17).

Using these definitions equation 3.9 can be simplified to

$$\tilde{\mathbf{G}} \oplus \tilde{\mathbf{F}} = (1 - \beta)\tilde{\mathbf{F}} + \tilde{\mathbf{G}}, \tag{3.15}$$

$$\gamma = (1 - \beta)\alpha + \beta, \tag{3.16}$$

where γ is the coverage value of $\tilde{\mathbf{G}} \oplus \tilde{\mathbf{F}}$.

Similar to the representation of unassociated pixels (equation 3.7) a premultiplied pixel has the following components

$$\tilde{\mathbf{F}} = \left(\tilde{r}, \tilde{g}, \tilde{b}, \alpha\right), \tag{3.17}$$

where $\tilde{r} = \alpha \cdot r$, $\tilde{g} = \alpha \cdot g$ and $\tilde{b} = \alpha \cdot b$. To retrieve the unassociated colours the components have to be divided by alpha

$$\mathbf{F} = \left(\frac{\tilde{r}}{\alpha}, \frac{\tilde{g}}{\alpha}, \frac{\tilde{b}}{\alpha}, \alpha\right), \tag{3.18}$$

where $\alpha \neq 0$. Table 3.1 shows the associated colours from figure 3.5. The premultiplied quadruple with alpha of 0 forces the colour components to result in black, as shown in table 3.1, the colour values of $\tilde{\mathbf{F}}_3$ and $\tilde{\mathbf{G}}_3$ are zeros. In comparison, there are nonzero colour values in $\mathbf{F}_3$ and $\mathbf{G}_3$ in figure 3.5. Moreover, an opaqueness approaching 0 reduces the precision of the colour components; especially when dealing with integer values. When calculating with floating precision, such as on modern graphics hardware, this is a minor issue. Colour values bigger than α indicate colours outside of the range $[0, 1]$, which is unusual and might apply for HDR images.

One reason for the usage of associated colour values, besides convenience and sparing one multiplication, can be shown when filtering images. In [12, chapter 16] it is shown that when images are *downsampled* and *composed*,

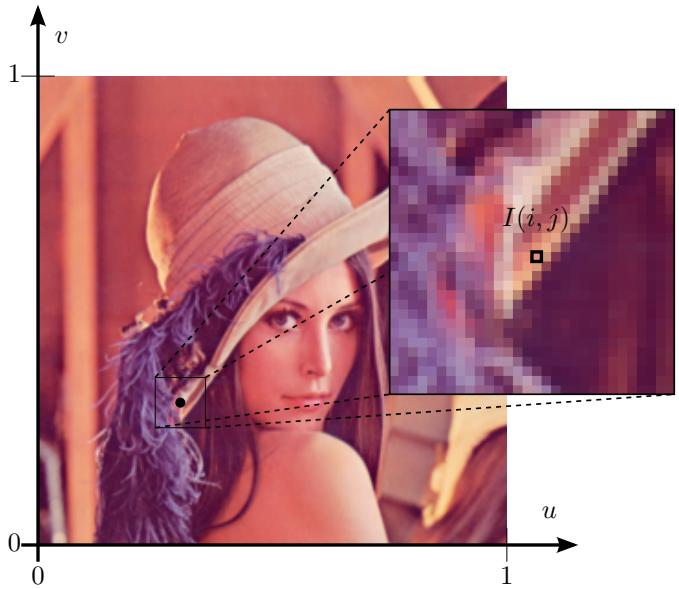


Figure 3.6: An image I used as texture. Accessing a pixel at the texture coordinates (u, v) is mapped to a discrete position (i, j) and the pixel $I(i, j)$ is returned.

premultiplied colours should be used to get results independent of the order of those operations. The result of downsampling two already blended images plus the result of downsampling those images and blending them afterwards is only the same when using premultiplied values. More details on downsampling and filtering images can be found in the next section.

3.4 Texture filtering

Texture coordinates (u, v) are continuous values in the range $[0, 1]$. This section explains the relation of u, v coordinates to textures, which are discrete functions, and how filtering operations are used when working with textures.

A texture I is used on a screen spaced method as discussed in section 3.2. In a certain fragment[7] the u, v coordinates obtained might be $(0.32, 0.29)$. When the texture's resolution is 256×256, the u, v coordinates have to be multiplied by 256 resulting in $(81.92, 74.24)$. Dropping the fractions, the corresponding colour value used in the fragment is $I(81, 74)$. This method is a form of the nearest neighbour filtering discussed in section 3.4.2. This example is taken from [2] and illustrated in figure 3.6. If the texture has the same pixel density as the projection on the screen, a simple mapping

[7] In computer graphics *fragments* represent pixels in a buffer not or not yet displayed on the screen, thus a fragment's content is changeable.

like the one previously described is applicable. However, there are situations where other algorithms should be applied. If the projected quadruple covers more pixels as the original texture contains, *magnification* (section 3.4.2) is needed. When the projection covers less pixels, *minification* (section 3.4.3) is needed. The goal is to prevent *aliasing* (section 3.4.4).

3.4.1 Convolution

Filtering/interpolating images, as discussed in sections 3.4.2 and 3.4.3, can be described by an operation called *linear convolution*. The term convolution, in mathematics, describes the combination of two functions of the same dimensionality either continuous or discrete. A convolution of a discrete function $I(i,j)$ is defined as

$$I'(u,v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j) \cdot H(u-i, v-j) \tag{3.19}$$

or

$$I' = I * H \tag{3.20}$$

where I is a discrete two-dimensional function, such as an image, and H is a two-dimensional continuous function (called filter kernel in image processing). If the filter kernel H is non-zero only in a certain range, i.e., within a width of w_H and a height of h_H, equation 3.19 can be changed to

$$I'(u,v) = \sum_{\substack{i= \\ \lfloor u-\frac{w_H}{2}+1 \rfloor}}^{\lfloor u+\frac{w_H}{2} \rfloor} \sum_{\substack{j= \\ \lfloor v-\frac{h_H}{2}+1 \rfloor}}^{\lfloor v+\frac{h_H}{2} \rfloor} I(i,j) \cdot H(u-i, v-j), \tag{3.21}$$

resulting in $w_H \cdot h_H \cdot w_I \cdot h_I$ operations where w_I and h_I are the dimensions of I.

Multiple channels

In image processing and computer graphics, images usually consist of more than one channel, such as colours, alpha or depth etc. When dealing with colour images, a pixel $\mathbf{I}(i,j)$ is multidimensional

$$\mathbf{I}(i,j) = \Big(I_r(i,j),\ I_g(i,j),\ I_b(i,j),\ I_\alpha(i,j) \Big). \tag{3.22}$$

Thus each channel c is convolved separately as

$$I'_c = I_c * H. \tag{3.23}$$

Separability

Some filter functions used for convolution are separable. Separability means that a two-dimensional function can be split up in a convolution of two one-dimensional functions. Thus, a filter kernel H can be composed of two kernels $H_x^{\rightarrow}, H_y^{\uparrow}$ as

$$H = H_x^{\rightarrow} * H_y^{\uparrow}. \tag{3.24}$$

With separable kernels equation 3.19 can be simplified to

$$I'(u,v) = \sum_{i=-\infty}^{\infty} H_x^{\rightarrow}(u-i) \cdot \sum_{j=-\infty}^{\infty} I(i,j) \cdot H_y^{\uparrow}(v-j). \tag{3.25}$$

Separable filters are much more efficient to calculate, especially with big filter kernels. Filtering with direct convolution (equation 3.19) results in a time complexity[8] of $\mathcal{O}(m^4)$. While filtering with separable kernels results in a complexity of $\mathcal{O}(m^3)$. An example of a separable filter is the Gaussian filter.

Gaussian filter

One frequently used filter, also used in this work, is the Gaussian filter, defined by

$$H_\sigma(r) = e^{-\frac{r^2}{2\sigma^2}} \qquad \text{or} \qquad H_\sigma(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{3.26}$$

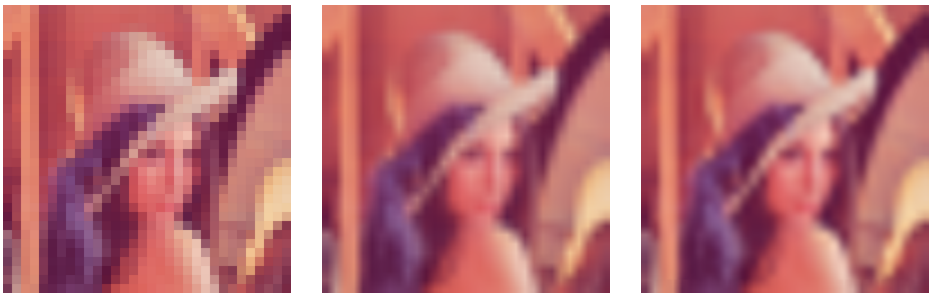
where σ is the standard deviation of the Gauss shaped function and r is the distance to the centre pixel [16]. A plot of a one-dimensional Gaussian function is shown in figure 3.9 (d). The Gaussian kernel is a separable filter; therefore

$$H_\sigma(x,y) = H_\sigma^{\rightarrow}(x) * H_\sigma^{\uparrow}(y). \tag{3.27}$$

3.4.2 Texture magnification

If a texture displayed on the screen has a lower resolution as the screen, it needs to be up-sampled or *magnified*. In this section, three methods are discussed with focus on methods used in computer graphics: nearest neighbour, bilinear and bicubic filtering [2].

[8] The term "complexity" describes the effort (i.e., computing time or storage) required by an algorithm or procedure to solve a particular problem in relation to the "problem size" m, [16].



(a) (b) (c)

Figure 3.7: Magnification of a 32×32 image onto 256×256 pixels. Filtering with the nearest neighbour filtering, where the nearest texel is chosen (a). Bilinear interpolation using a weighted average of the four nearest pixels (b). Bicubic filtering as implemented in ImageJ [http://rsbweb.nih.gov/ij/features.html] using an weighted average of the 4×4 nearest pixels (c), as discussed in [16, chapter 16].

Nearest neighbour filtering

With *nearest neighbour filtering* the nearest texel to a sample position in the source image is used. Basically, this is the method described with an example at the introduction of section 3.4. This interpolation method might also be refereed as *pixelation* or *box filtering* (although box filtering usually refers to broader filters). The results of this method are not visually appealing as shown in figure 3.7 (a). However, it is one of the fastest interpolation methods.

The nearest neighbour interpolation or, generally speaking, the *box* filter can be expressed as convolution filter such as

$$H_{\text{box}}(x, w_{\text{box}}) = \begin{cases} 1 & \text{for } -\frac{w_{\text{box}}}{2} \leq |x| < \frac{w_{\text{box}}}{2}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.28}$$

where the *nearest neighbour filter* is a special case of the box filter:

$$H_{\text{nn}}(x) = H_{\text{box}}(x, 1). \tag{3.29}$$

Note that the above formulations are one-dimensional filter kernels. Figure 3.9 (a) shows a plot of the nearest neighbour interpolation function.

Bilinear interpolation

Figure 3.7 (b) shows a *bilinear* interpolation sometimes referred to as *linear* interpolation. For this type of filter four neighbouring pixels are linearly interpolated and added together. Continuing the example from the introduction of section 3.4, the texel-coordinates obtained are $(u, v) = (81.92, 74.24)$.

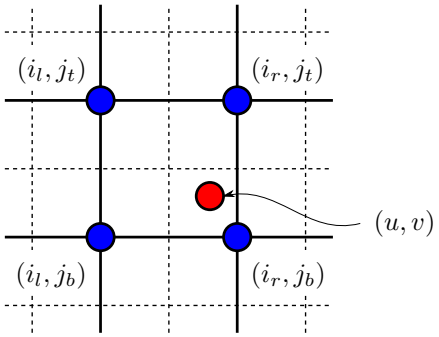


Figure 3.8: Notation for bilinear interpolation, where the four texels are illustrated by the four squares drawn by the dashed lines. The four points illustrate the texel centers at $(i_l, j_t), (i_l, j_b), (i_r, j_t)$ and (i_r, j_b), which are the nearest neighbours to the point at (u, v) (inspired by [2]).

Computing the four closest pixels retrieves the rectangle area ranging from $(i_l, j_b) = (81, 74)$ to $(i_r, j_t) = (82, 75)$. For the interpolation, the decimal part of the (u, v) coordinates are needed $(u', v') = (u - \lfloor u \rfloor, v - \lfloor v \rfloor)$. The bilinearly interpolated value I_{bilinear} at the position (u, v) is defined as

$$\begin{aligned} I_{\text{bilinear}}(u, v) = {} & I(i_l, j_b) \cdot (1 - u') \cdot (1 - v') + I(i_r, j_b) \cdot u' \cdot (1 - v') \\ & + I(i_l, j_t) \cdot (1 - u') \cdot v' + I(i_r, j_t) \cdot u' \cdot v', \end{aligned} \tag{3.30}$$

where $I(i, j)$ is the texels' value at the integer position (i, j) as shown in figure 3.8. Bilinear filtering is widely used in computer graphics. Especially because of its simplicity and superior results compared to nearest neighbour filtering (figure 3.7 (b)). The bilinear interpolation can also be expressed as a filter kernel for convolution by the equation

$$H_{\text{bilinear}}(u, v) = H^{\rightarrow}_{\text{linear}}(u) * H^{\uparrow}_{\text{linear}}(v), \tag{3.31}$$

$$H^{\rightarrow}_{linear}(r) = H^{\uparrow}_{linear}(r) = \begin{cases} 1 - |r| & \text{for } |r| < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.32}$$

Figure 3.9 (b) shows a plot of the one-dimensional function $H^{\rightarrow}_{linear}$.

Bicubic filtering

Cubic interpolation uses a weighted sum of an array of texels thereby using more data for the interpolation. This results in a higher quality than box or bilinear filtering but also in a more complex calculation. One example cubic

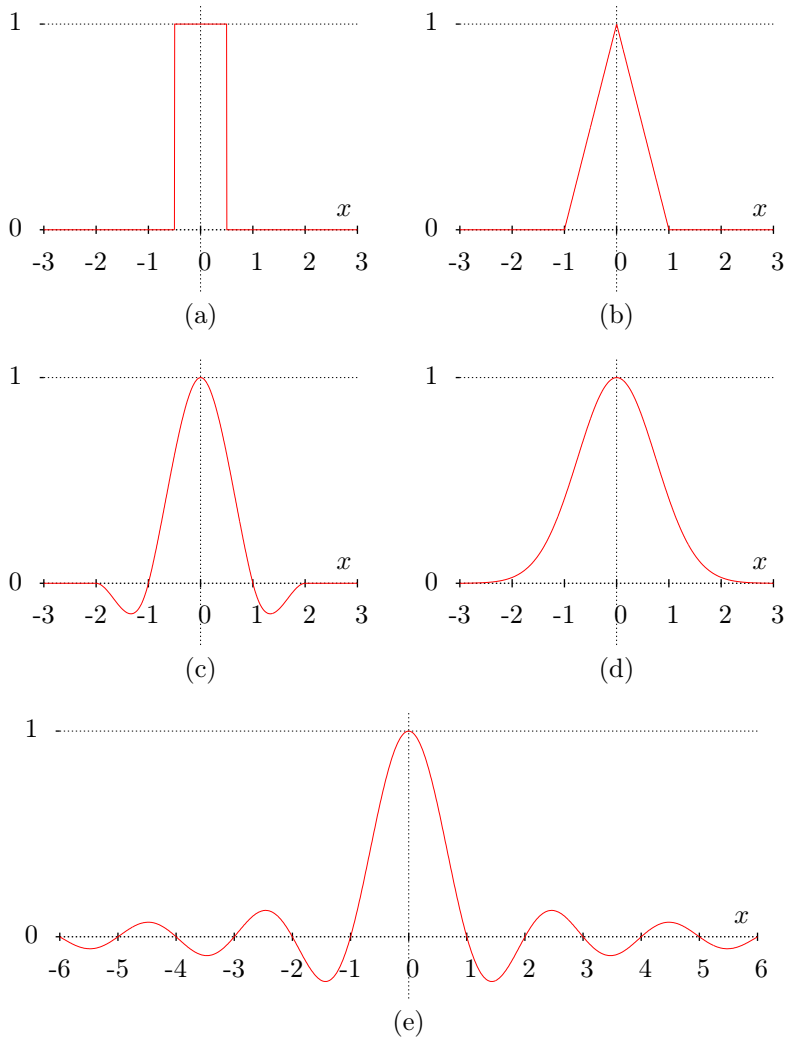


Figure 3.9: The interpolation kernel of a *nearest neighbour* interpolation filter from equation 3.29 (a), of an *linear* interpolation as in equation 3.32 (b), and a *cubic* interpolation filter with the steepness value $a = 1.0$ calculated with equation 3.33 (c). A one-dimensional Gaussian function with $\sigma = 0.75$ from equation 3.26 (d), and a plot of the infinitely defined *sinc* function as in equation 3.35 (e).

function H_{cubic} is defined as

$$H_{\text{cubic}}(x) = \begin{cases} (-a+2) \cdot |x|^3 + (a-3) \cdot |x|^2 + 1 & \text{for } 0 \leq |x| \leq 1, \\ -a \cdot |x|^3 + 5a \cdot |x|^2 - 8a \cdot |x| + 4a & \text{for } 1 < |x| < 2, \\ 0 & \text{otherwise,} \end{cases} \tag{3.33}$$

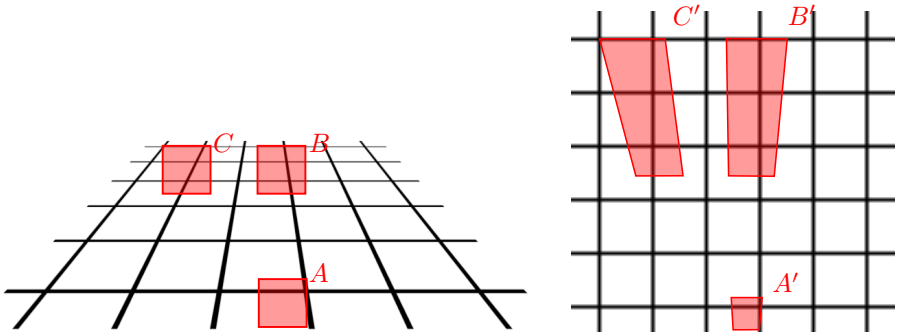


Figure 3.10: Square pixel cells A, B, C in screen space (a) and their projections A', B', C' onto a texture (b). While the projection of A covers a small area in texture space A', the projections B' and C' cover large areas.

with a defining the steepness of the slope [16]. A standard value of $a = 1$ is often recommended. The bicubic interpolated value $I_{\text{bicubic}}(u, v)$ is defined by the convolution

$$I_{\text{bicubic}}(u,v) = \sum_{\substack{i= \\ \lfloor u \rfloor -1}}^{\lfloor u \rfloor +2} \sum_{\substack{j= \\ \lfloor v \rfloor -1}}^{\lfloor v \rfloor +2} I(i,j) \cdot H_{cubic}(u-i) \cdot H_{cubic}(v-j). \tag{3.34}$$

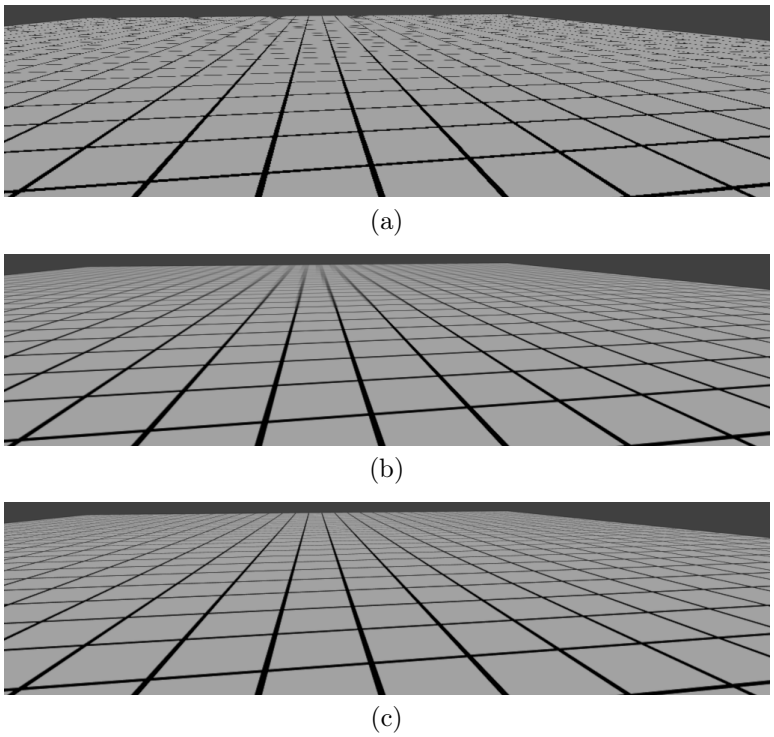
The result of a bicubic interpolation is shown in figure 3.7 (c) and a plot of H_{cubic} is shown in figure 3.9 (c).

3.4.3 Texture minification

The magnification of a low-resolution texture, as discussed in section 3.4.2, cannot have the same visual quality as a high-resolution texture for obvious reasons. The opposite, creating a low-resolution texel of a high-resolution texture, needs different considerations. The ideal solution is [2]:

> To get a correct colour value for each pixel, you should integrate the effect of the texels influencing the pixel. However, it is difficult to determine precisely the exact influence of all texels near a particular pixel, and it is effectively impossible to do so perfectly in real time.

For minification, the same filters as used for magnification can be used. Those filters fail if the pixel's area covers a relatively big area of texels. This effect is shown in figure 3.10 (a) where the pixels B and C cover a



(a) (b) (c)

Figure 3.11: Images Rendered with nearest neighbour sampling (a), mipmapping (b) and with summed area tables (c), as implemented in Blender (www.blender.org).

big area of texels after projecting them to B' and C' (figure 3.10 (b)). The nearest neighbour interpolation uses the colour information of the nearest texel, resulting in visible artefacts. Blinking artefacts appear especially if the texture in the scene is moving. Bilinear filtering uses the four nearest texels and fails to achieve artefact free results if the covered area of the pixel is bigger than four texels. The same applies for the area of 4×4 texels with bicubic sampling. Nearest neighbour sampling produces the most noticeable artefacts. In figure 3.11 (a) nearest neighbour filtering is shown along other methods to reduce those artefacts. This problem is called *aliasing* and can be explained with the Nyquist-Shannon sampling theorem (discussed in the next section).

3.4.4 Aliasing

Often continuous signals, such as audio or video signals, are sampled at uniformly spaced intervals resulting in them being discretised. This is done for digitally representing the signal and to reduce the amount of information. Whenever the process of *sampling* is done, aliasing may occur. One classic,

real life example is a spinning wheel filmed by a movie camera. Because the wheel might spin faster than the camera can record an image, it might appear to be spinning slowly forward, backward, or even stand still.

In computer graphics, common examples of aliasing are the "jaggies" of a rasterised line or the minification of a checker or line pattern (shown in figure 3.11 (a)) [2]. For a signal to be sampled properly, the sampling frequency has to be more than twice the maximum frequency of the signal to be sampled. This is called the Nyquist-Shannon theorem also widely known as the "sampling theorem". In the real world example of the spinning wheel, this would mean that a movie camera has to shoot at least more than two pictures while the wheel is turning once.

The sampling theorem also applies to 2-D functions such as images. Therefore, aliasing occurs when a texture is sampled with a sampling frequency θ_s where the maximum signal frequency θ_{max} is

$$2\theta_{max} > \theta_s.$$

Figure 3.11 (a) shows a rendering with aliasing artefacts. The options to fulfil the sampling theorem are

- to increase the sampling frequency θ_s until $2\theta_{max} \leq \theta_s$,
- or to reduce the bandwidth (the maximum frequency) in the function.

While increasing the sampling frequency obviously leads to more computation costs, depending on the sampled area, the latter option seems reasonable. To reduce the maximum frequency (band limit) in a function, the function should be filtered by an arbitrary low-pass filter.[9]

The ideal low-pass filter is the sinc function

$$\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}, \tag{3.35}$$

as ploted in figure 3.9 (e). However, the filter width of the sinc function is infinite defined thus making it computationally expensive to compute and rarely useful.

In computer graphics, there are many attempts to overcome aliasing artefacts and some are discussed in the following sections. In general, they are referred to as *antialiasing* methods. Blinn discussed this topic in [12, chapter 3] and his rather demotivating verdict is:

> There is no such thing as full antialiasing. Anyone who tries to tell you differently is trying to sell you something.

There are various points in a rendering process where aliasing can occur. Drawing lines or edges in the frame buffer is a well known source of aliasing.

[9] The term low-pass filter describes the ability of a filter to remove high frequencies and keep low frequencies in the signal.

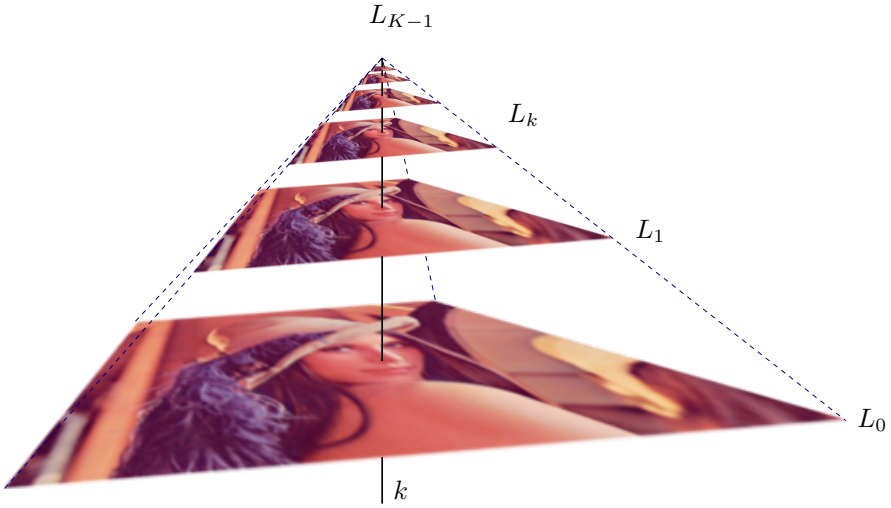


Figure 3.12: A *mipmap* with its original image L_0, at the base. The next level up is formed by averaging each 2×2 area into one texel of the coarser level. The vertical axis k can be seen as third texture coordinate, thus mipmapping is also called *trilinear interpolation* (adapted from [2]).

Methods to overcome such aliasing artefacts are e.g., multisample antialiasing (MSAA) or post processing antialiasing methods such as morphological antialiasing (MLAA) [65].

In the following sections, filtering methods used in computer graphics to overcome aliasing in textures are discussed. General surveys and summaries of filtering functions can be found in [36, 37].

Mipmapping

One technique frequently used in computer graphics to reduce aliasing is mipmapping [13, 14]:

> When mipmapping, we build scaled-down versions of our texture maps; when rendering portions of the scene where low texture detail is needed, we use the smaller textures. Mipmapping can save memory and rendering time, but the motivating idea behind the technique's initial formulation was to increase the quality of the scene by reducing aliasing.

The general idea is to use K textures of different scaling levels (level 0 to $K-1$) by generating a pyramid, as illustrated in figure 3.12. The coarser levels are generated by downsampling the previous level by 0.5×0.5 of its resolution, with a 2×2 box-filter (see section 3.4.2). Mipmapping has its origin in [73] and is the reason why square textures with power-of-two resolutions are used in computer graphics. Although the box-filter is often used

(a reason might be its simplicity and efficiency), it can be shown that averaging produces artefacts [12, chapter 3]. The best filter would be the sinc function (equation 3.35), but it is defined infinitely. Finite filters with good results in reducing aliasing artefacts are the Lanczos-windowed sinc function or the Kaiser-windowed sinc function [13].

For accessing the mipmap structure while texturing, the area of a projected screen pixel has to be determined (shown in figure 3.10 (b)). With the area, the corresponding mipmap level k_{mip} can be determined as illustrated in figure 3.13. Two methods for computing k_{mip} are to use the longer edge of the quadrilateral of the pixel's cell or the largest absolute value of the differentials of the texture coordinates (u, v), each in the directions x and y.

The goal is a pixel-to-texel ratio of at least 1:1 in order to achieve the Nyquist rate. The mipmap is accessed by the triplet (u, v, k) whereas the colour is linearly interpolated from two sample points on the k-axis. Therefore, this process is called *trilinear interpolation*.

The advantage of mipmaps is that, no matter what the amount of minification is, the sampling process takes a fixed amount of time. However, one major flaw of mipmapping is overblurring which occurs if a pixel covers a large area of texels only in one direction. For example, at the lines moving into distance shown in figure 3.11 (b).

Ripmaps are an extension to mipmapping and reduce the overblurring problem by storing all the possible rectangles 1×2, 2×4, 2×1, 4×1, etc. additionally to the rectangles of a mipmap $1 \times 1, 2 \times 2, 4 \times 4$, etc. Ripmaps produce better visuals than mipmaps but come at a high cost by adding three times more additional space beyond the original image whereas mipmaps only add one-third storage space [2].

Anisotropic filtering

Anisotropic texture filtering, which reuses mipmapping, can be used for better results on widely spread graphics hardware. The idea is to sample the texture a number of times depending on the pixel's projection on the texture. Each tap is a sample of a hirachical mipmap pyramid (see [3, 31, 53]). The samples are taken along the lines of anisotropy and afterwards averaged. Thus, anisotropic filtering can be implemented in programmable shaders by simple texture lookups. This is roughly illustrated in figure 3.13. In mipmapping, the level of detail k is defined by the longest side of a pixel's projection (k_{mip}). Whereas with anisotropic filtering, it is defined by the shortest one (k_{ani}). This makes the averaged area smaller for each sample and therefore less blurrier. The overblurring artefacts known from general mipmapping can be avoided with this technique. The visual improvement comes with additional sampling costs. On modern GPUs the implementation of anisotropic filtering varies. The OpenGL extension specification does not specify any par-

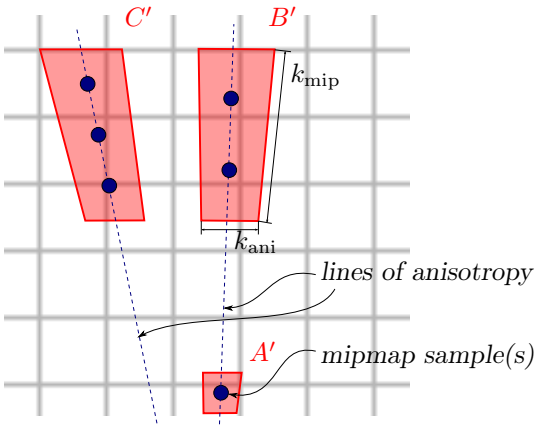


Figure 3.13: An extension of figure 3.10 (b) showing the lines of anisotropy used for anisotropic filtering. Three projected pixels A', B', C' on the texture are sampled with $1, 2$ and 3 mipmap samples per pixel. The samples are taken on the line of anisotropy. Additionally the lengths of the quadrilateral k_{mip} and k_{ani} used for accessing the mipmap structure in mipmapping and anisotropic filtering, respectively, is shown (adapted from [2]).

ticular implementation.[10] Therefore, the number of samples are not strictly defined and are often configurable. Using more samples results in higher quality and more computing time.

Recursive filtering

Instead of directly filtering with large filters smaller filters can be applied recursively. The notation for recursively applying a box filter is H_{box}^{*m} where H_{box} is the kernel of a filter and ^{*m} denotes m convolutions:

$$H_{\text{box}}^{*m} = H_{\text{box}} * H_{\text{box}} * \cdots * H_{\text{box}} \qquad (m \text{ times}).$$

Figures 3.14 (b)–(e) show plots of the impulse response of recursive box filters. As m increases the recursive filter approaches a Gaussian shape [35]. In comparison, figures 3.14 (f) and (g) show the impulse response of Gaussian filters.

If Gaussian filters, of sizes σ_1, σ_2, are applied recursively to an image, the result is the same as filtering the image with one bigger Gaussian kernel H_{σ_c}. Thus, H_{σ_c} can be composed by

$$H_{\sigma_c} = H_{\sigma_1} * H_{\sigma_2}, \tag{3.36}$$

[10] http://www.opengl.org/registry/specs/EXT/texture_filter_anisotropic.txt

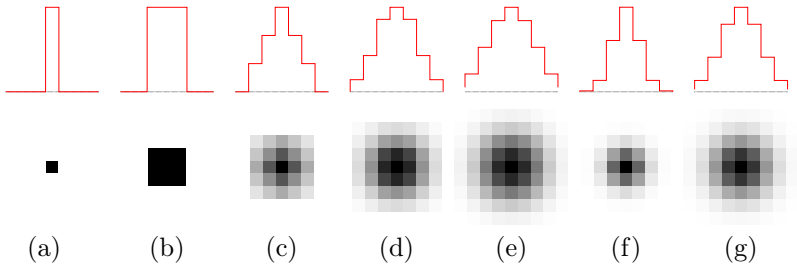


Figure 3.14: The original signal is a 9×9 pixel image with an impulse at $(4, 4)$ (a). The signal convolved with recursive box filtering of the type $H_{\text{box}}, H^{*2}_{\text{box}}, H^{*3}_{\text{box}}$ and H^{*4}_{box} (b-e), respectively. For comparison the convolutions of (a) with a Gauss filters of sizes $\sigma = 1.0$ and $\sigma = 1.5$ are shown (f) and (g). Note that all plots and images are normalised so that the maximum value is 1 for a better visualization. The plots show the 4^{th} column of the convolved images.

where $\sigma_c = \sqrt{\sigma_1^2 + \sigma_2^2}$ is the size of the combined filter [17, chapter 8].

Strengert et al. in [72] use recursive filtering on pyramid levels to efficiently approximate higher order filters, i.e., Gaussian filters and B-splines. The authors use bilinear interpolation of GPUs to speed up the calculations. A more thorough exploration can be found in [45]. The authors show that filters such as Bartlett or Gauss can be approximated by

$$H^{7\times7}_{Gauss} = H^{3\times3}_{Bartlett} * H^{3\times3}_{Bartlett} * H^{3\times3}_{Bartlett}, \tag{3.37}$$

$$H^{3\times3}_{Bartlett} = H^{\searrow}_{\text{box}} * H^{\nwarrow}_{\text{box}}, \tag{3.38}$$

where $H^{\searrow}_{\text{box}}$ and $H^{\nwarrow}_{\text{box}}$ are shifted kernels of 2×2 box filters:

$$H^{\searrow}_{\text{box}} = \frac{1}{4}\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \qquad H^{\nwarrow}_{\text{box}} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3.39}$$

$$\tag{3.40}$$

In [45] the authors extend this approach by introducing *analysis* and *synthesis* filters. While analysis filters are used to downsample, synthesis filters are used to upsample an image by factor (0.5×0.5) and (2×2) respectively. The notation for analysis filters is denoted by

$$I' = I \downarrow *H, \tag{3.41}$$

where I is the input image, I' the downsampled image, and H the filter kernel used for downsampling. The authors propose discontinuous and continuous filters composeable with the kernels shown in equations 3.39 and 3.38.

Synthesis filters are denoted by

$$I' = I \uparrow *H. \tag{3.42}$$

For the upsampling process, the authors propose the use of B-splines as described in [18].

In [46] the synthesis and analysis approach for efficiently blurring renderings is used in a DoF method (see section 4.2.3). However, this filtering approach produces artefacts when used in animated scenes. Therefore in [44] the authors do a quantitative analysis of the filters discussed in [72] and propose an improved analysis filter.

Summed area tables

Crow provided a method by which box filtering of an image over any aligned rectangle can be done rapidly. This method is called *summed area tables* and got introduced in [21]. The method takes a source image I and creates a new image, S, whose value at pixel (x, y) is a sum of all values of the rectangle with corners $(0, 0)$ and (x, y) in image I

$$S(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j), \tag{3.43}$$

where $x, y, i, j \in \mathbb{N}_0$. The sum of a rectangle with the corners at (x, y) and $(x + a, y + b)$ can then be computed by

$$\sum_{i=x}^{x+a} \sum_{j=y}^{y+b} I(i, j) = S(x + a, y + b) - S(x + a, y) - S(x, y + b) + S(x, y). \tag{3.44}$$

The advantage of this method is that once the summed area table S is computed, filtering with a box-filter of any size is possible within constant time by doing four texture lookups and the arithmetic operations in equation 3.44. Note that S needs more bits of precision to store the sum of colour values (e.g., 16 bits or more for each channel). Further information on summed area tables can be found in [29].

A method to produce a summed area table on modern GPUs is described in [38]. In figure 3.11 (c) a summed area table is applied and shows visual improvements over nearest neighbour filtering and mipmapping, figure 3.11 (a) and (b) respectively. If a transformed pixel is not approximately an aligned rectangle, then summed areas might blur the result excessively.

Heckbert improved summed area tables by proposing a generalized version of summed area tables where repeated box filters, converging toward a Gaussian shape, and other polynomial filter kernels can be used for convolution [35]. The basic idea is to take advantage of the fact that the convolution

of f with g is equivalent to convolution of the m^{th} integral of f with the m^{th} derivative of g

$$f * g = f_m * g_{-m}, \tag{3.45}$$

where m (positive) denotes integration and $-m$ (negative) denotes differentiation. Preprocessing and the filtering process are computationally more expensive than simple summed area tables.

Heat diffusion

One approach for blurring an image inhomogeneously is to use heat diffusion. A heat equation simulates the thermal conductivity of a medium I (in this case an image) over time t and is defined as:

$$\frac{\delta I}{\delta t} = \nabla \cdot (g \nabla I), \tag{3.46}$$

where ∇f of the function f donates the first derivative the function (f') and g is the conductivity, a function which controls the diffusion. The function g can be defined as gradient of I to implement and edge preserving filter [17, chapter 3] or use any other function, such as a CoC-gradient for DoF simulations [9, 39]. To allow smoothing in certain areas, g should return high values. In areas where smoothing should be suppressed, g should return low values. For further information one might refer to [58]. The conductivity is often in the range $[0, 1]$ and if g is not uniform, anisotropic filtering is done.

If the conductivity is constant, equation 3.46 reduces to

$$\frac{\delta I}{\delta t} = c \cdot (\nabla^2 I), \tag{3.47}$$

where c is the constant conductivity and $\nabla^2 I$ is the *Laplace operator*[11] of the image I. The solution for equation 3.47 can be calculated numerically. The heat equation is solved iteratively in the form

$$I_m = \begin{cases} I_0 & \text{for } m = 0, \\ I_{m-1} + \alpha \cdot [\nabla^2 I_{m-1}] & \text{for } m > 0, \end{cases} \tag{3.48}$$

where I_0 is the original signal and α is a time increment, which should be in the range $[0, 0.25]$. The Laplacian ($\nabla^2 I$) in equation 3.48 can be approximated by the convolution

$$\nabla^2 I \approx I * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \tag{3.49}$$

[11] The Laplace operator or Laplacian $\nabla^2 f$ is a differential operator given by the divergence of the gradient of function f. Other notations for the Laplacian of f are $\nabla \cdot \nabla f$, $\nabla^2 f$ or Δf http://en.wikipedia.org/wiki/Laplace_operator.

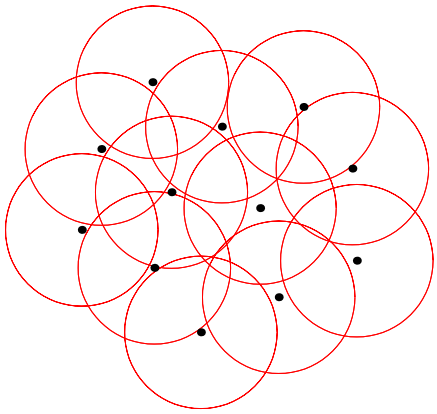


Figure 3.15: A Poisson disk generated with Poisson Disk Generator (http://www.coderhaus.com/?p=11). The circles around the points mark the minimum distance $r = 0.1$.

for a discrete image I. After m diffusion steps, image I_m is the same as filtering I_0 with a normalised Gaussian kernel of width $\sigma = \sqrt{2m\alpha}$ [17, chapter 3]. More on the similarities between heat diffusion and the Gaussian distribution can be found in [55].

Stochastic sampling

One method to overcome aliasing issues in sampling is to skip *uniform sampling* (i.e., mipmapping, summed area tables, ...) and use *stochastic sampling*. Stochastic sampling is often used in ray tracing to distribute rays [19], but it is also applicable in other fields. In [23] the authors state that:

> Both the grain of photographic film and the receptor patterns of the human retina exhibit random sampling.

The advantage of stochastic sampling over uniform sampling is the fact that high frequency information is scattered into broadband noise rather than generating false patterns. Thus, stochastic sampling can be used in many situations. One example is [68], discussed in section 4.2, where Poisson disk sampling is used in combination with trilinear filtering. Two popular stochastic sampling methods are *Poisson* and *jittered* patterns.

Poisson sampling: The basic idea of generating a Poisson disk sampling pattern is to randomly generate sample points with the constraint that there is a minimum distance r between sample points (see figure 3.15). There are several implementations of this approach. One efficient implementation is proposed in [15]. Although there are methods to generate Poisson patterns

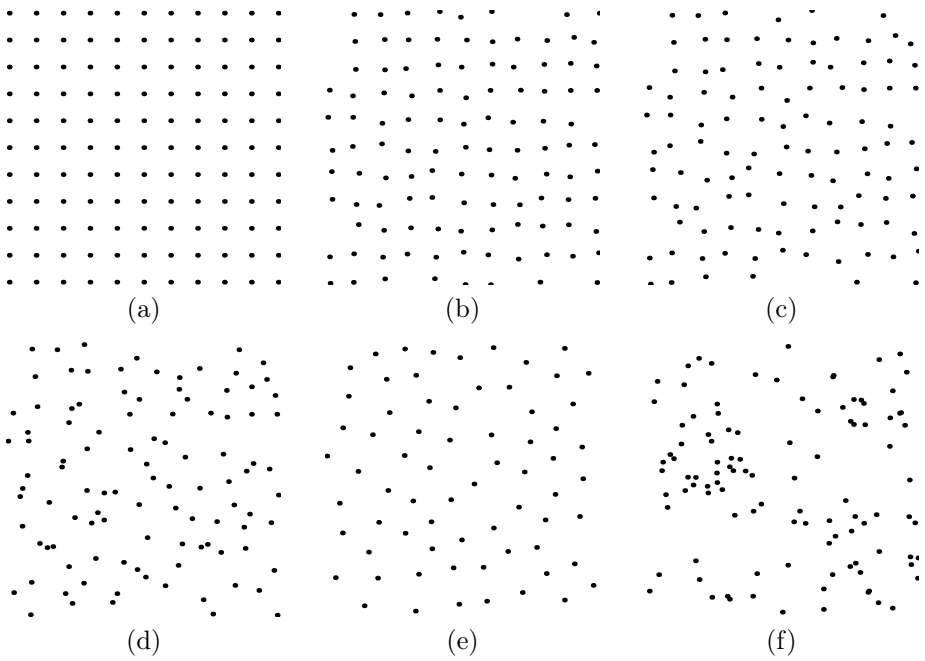


Figure 3.16: Stochastic sampling patterns. A uniform grid with a distance of 0.1 between the points (a). Rectangular jittered sampling with jitter distances (distances from the grid position) of 0.3 (b), 0.5 (c) and 1 (d). Poisson disk distribution with a minimum distance of 0.1 (e), generated with Poisson Disk Generator (http://www.coderhaus.com/?p=11) discussed in [15]. A stochastic pattern generated by randomly splatting points (f).

efficiently, the generation is a costly process. Therefore in computer graphics, and some other applications, the Poisson pattern is stored in a LUT to avoid the generation process during run-time.

A Poisson sampling pattern is shown in figure 3.16 (e). Due to the minimum distance between each sample point, high densities of spots in certain areas are avoided, compared to randomly generated sample points in figure 3.16 (f). By increasing the minimum distance in Poisson sampling, high-frequency noise is increased and low-frequency noise is reduced [19].

Jittered sampling: Another approach to generate a stochastic sample pattern is jittering. A regular sampling pattern, as shown in figure 3.16 (a), is modified by jittering each sample point. The regular sample point y_k (with sampling rate β) is modified by adding a random variable j_k

$$y_k = \frac{k}{\beta} \qquad\qquad x_k = y_k + j_k \tag{3.50}$$

where k is in the range $k = [-\infty, \infty]$. The random number j_k is in the range $j_k = [-\alpha/(2\beta), \alpha/(2\beta)]$. Values of $\alpha = 0.3, 0.5$ or 1.0 are shown in

figures 3.16 (b)–(d). An $\alpha = 0$ produces the same results as uniform sampling (see figure 3.16 (a)). Compared to Poisson sampling, high-frequency noise is decreased but aliasing appears at jittering rates of $\alpha < 1.0$. Jittered sampling is a good approximation of Poisson sampling and can be calculated more efficiently.

However, according to [19]:

> ... images are somewhat noisier and some very small amount of aliasing can remain.

Thus, Poisson sampling seems to produce better results when jittered and Poisson sampling are compared.

Chapter 4

Previous work

The optical principles of depth of field have been discussed previously in section 2.3. Methods for simulating DoF in computer graphics are discussed in the following sections. Some DoF implementations, especially rasterisation techniques, sacrifice quality for high frame rates and produce artefacts such as discussed in section 4.2.1. In this work DoF simulations are structured in *object spaced* (section 4.1) and *image spaced* methods (section 4.2). Object spaced methods operate in 3D space and use the rendering pipeline for the DoF effect whereas image spaced methods operate on a rendered image; the latter is known as post processing and explained in section 3.2. In general, object-space methods produce more realistic results than image spaced methods. Despite that, the latter are faster. Summary and surveys of depth of field methods can be found in [1, 2, 4–6, 22].

4.1 Methods in object space

DoF methods in this work are classified as object spaced methods if the DoF effect is generated by altering the rendering method, i.e., changing rays in a ray tracer or modifying the position of a camera while rendering. While image spaced methods do not (or only slightly) modify the rendering method. In the following sections object spaced methods are discussed.

4.1.1 Distributed ray tracing

DoF effects can be simulated by distributed ray tracing [20]. Instead of tracing one ray per pixel, which would simulate a pinhole camera, several rays are distributed per pixel position thus simulating a finite aperture. This technique is called *oversampling* and is used to reduce aliasing artefacts [5]. If the rays are distributed within a finite aperture, DoF can be simulated. Rays can be refracted by the lens and then enter the scene. Since this simulates the way how an image is formed in optical systems (such as cameras) im-

ages appear realistic. Therefore, ray tracing often serves as a reference for evaluating postprocess methods [4]. To get visual appealing DoF effects (especially for large blurs) many rays per pixel are required. Thus, making ray tracing a technique hardly usable for interactive applications. When fewer rays are used, the technique degrades to noise rather than to other visible artefacts. More details and implementation hints can be found in [60] which is a discussion of the source code for the ray tracer pbrt.[1] Furthermore, the ray-casting process of rays in pbrt is explained in detail in section 6.6.

4.1.2 Accumulation buffer

Standard rasterisation hardware, such as a modern GPU, is optimized for rendering pinhole models. Therefore, rasterisation methods are widely used in interactive applications, e.g., games. The accumulation buffer method uses pinhole renderings of a scene from different camera-locations, uniformly distributed across the aperture, and then blends those renderings together using the accumulation buffer [33]. This method is similar to the ray tracing approach proposed in [20] since many samples per pixel are taken and averaged. Although the accumulation buffer method is faster than distributed ray tracing, large blurs need more scene renderings to overcome banding artefacts. For a high quality rendering (limiting sampling artefacts to 2×2 pixel with an 8-pixel-radius CoC maximum) 50 renderings are required. For lower quality (limiting artefact to 3×3 pixels with a 6-pixel-radius CoC) 12 scene renderings are needed (according to [22]). This is still hardly applicable for interactive applications. Since rasterisation is used, with the accumulation buffer technique, the entire scene is rendered. Thus, it is not possible to adaptively reduce the rendering effort for pixels in focus with small or almost no blur. With ray-tracing techniques, this adaptive control is possible.

4.1.3 Splatting

Point based methods describe the scene as points rather than as geometric primitives. These points are rendered by elliptical weighting and painted as ellipses in the screen buffer. To simulate DoF, while rendering point based scenes, the ellipses can be scaled up to the CoC (discussed in section 2.3). Although DoF blurring naturally fits in the pipeline for point based renderings, e.g., [47], a similar method can also be applied to primitive based approaches [49]. To render points on rasterisation hardware, additional geometry, such as CoC sized sprites, has to be generated. These sprites can be generated with geometry shaders[2] [2]. With sprites, each point can spread its colour value based on a PSF (section 2.2). If a pixel's area is increased, the

[1] Source code available at http://www.pbrt.org/.

[2] Geometry shaders add a additional programmable stage between vertex shaders and the rasterisation unit on the GPU's rendering pipeline.

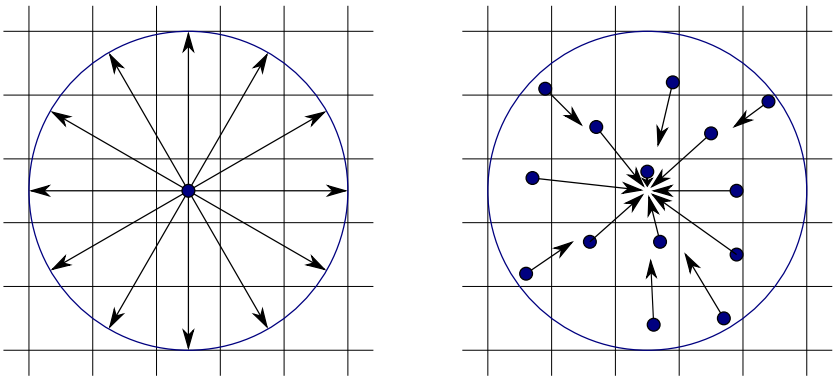


Figure 4.1: Scattering spreads a pixel's value to the neighbouring area (a). Gathering samples the neighbouring values to affect one pixel. The samples' positions might be defined by a Poisson disk (b). Programmable shaders on modern GPUs are optimized for gathering via texture sampling. (adapted from [2]).

sum of its intensity should stay the same. This is done by normalisation and is reflected in the alpha channel of the pixels. To compose the final scene, the sprites have to be rendered with enabled alpha-blending and should be rendered from back to front. The fact that a pixel value spreads its colour value is called *scattering* as illustrated in figure 4.1 (a). Spreading a colour value does not map well to pixel shader capabilities of GPUs. Additionally, generating geometry and depth-sorting (needed for artefact free spreading on GPUs) are costly operations. Scattering suffers from the lack of high-precision blending on GPUs and often normalisation issues occur. Therefore, splatting techniques are mostly used in non-interactive applications and often named *forward mapped methods* because of the fact that source pixels are mapped onto the destination image [22].

4.2 Image space methods

DoF methods in object space, as discussed in the previous section, are hardly applicable for real-time applications. In contrast, image spaced depth-of-field methods are widely used in real-time applications. Image spaced DoF methods are based on the idea of rendering the scene with a pinhole camera model and simulating the DoF effect via post processing. Thus, leading to few or no changes in the rendering pipeline. Potmesil and Chakravarty discussed such depth-of-field methods first in [62]. For each pixel, the CoC is calculated based on the depth of a pixel (see section 2.3). Pixels are blurred according to their CoC. Image spaced methods use additional buffers to store more in-

formation per pixel such as depth. Since image based DoF effects simulate an effect which usually happens within a scene but produce the effect afterwards, such methods are called *backwards mapped* or *reverse mapped* methods. Although post-processing methods work on an image, there are post-processing DoF techniques which slightly modify the way how the scene is rendered. One such possibility is to render into layers. Therefore, DoF methods can be classified into methods with one or more images (called *single-layer* or *multi-layer methods*). Most image-based DoF methods use *gathering* and not spreading because of performance reasons. Gathering means that a pixel samples neighbouring texels and can be carried out efficiently through texture lookups on GPUs. Figure 4.1 (b) outlines how gathering can be performed. Despite that, there are image based DoF methods which use *scattering*, e.g., [40, 49]. Thus, scattering and gathering can be used as another classification criterion for DoF methods.

Spreading or gathering

Another way to look at gathering and scattering can be found in [42], where the author explains linear image filters by order-4-tensors simplified by two-dimensional matrices. Considering an image to be a vector of pixels $\mathbf{I}$ a filtered image $\mathbf{I}'$ can be obtained by a matrix vector multiplication $\mathbf{I}' = \mathbf{A} \cdot \mathbf{I}$ where $\mathbf{A}$ is a linear filter in matrix form. Filter weights placed in the rows of $\mathbf{A}$ produce a gathering operation and arranging the weights down the column has the effect of spreading. With this notation it can be shown that some spreading methods produce the same results as gathering methods and some filter kernels can be applied in both ways. In [40] a DoF technique using such spreading filters is presented (described in section 4.2.2).

4.2.1 Artefacts

Artefacts in DoF methods occur due to the facts that image based DoF approaches do not resemble the transportation of light such as ray tracing; and image based DoF often prioritise speed instead of physical accuracy. Some of those artefacts are discussed in the following sections.

Intensity leakage

One artefact, in screen spaced approaches, is the so called *intensity leakage*. It happens when pixels are blurred out with an isotropic gathering blur method. Sharp pixels bleed their colour on unsharp pixels due to large CoCs of the latter. Thus, this artefact might be also called *pixel bleeding* or *colour bleeding*. Intensity leakage artefacts are often noticeable when in focus objects are in front of a blurred background. In figure 4.2 (b), one such artefact is shown.

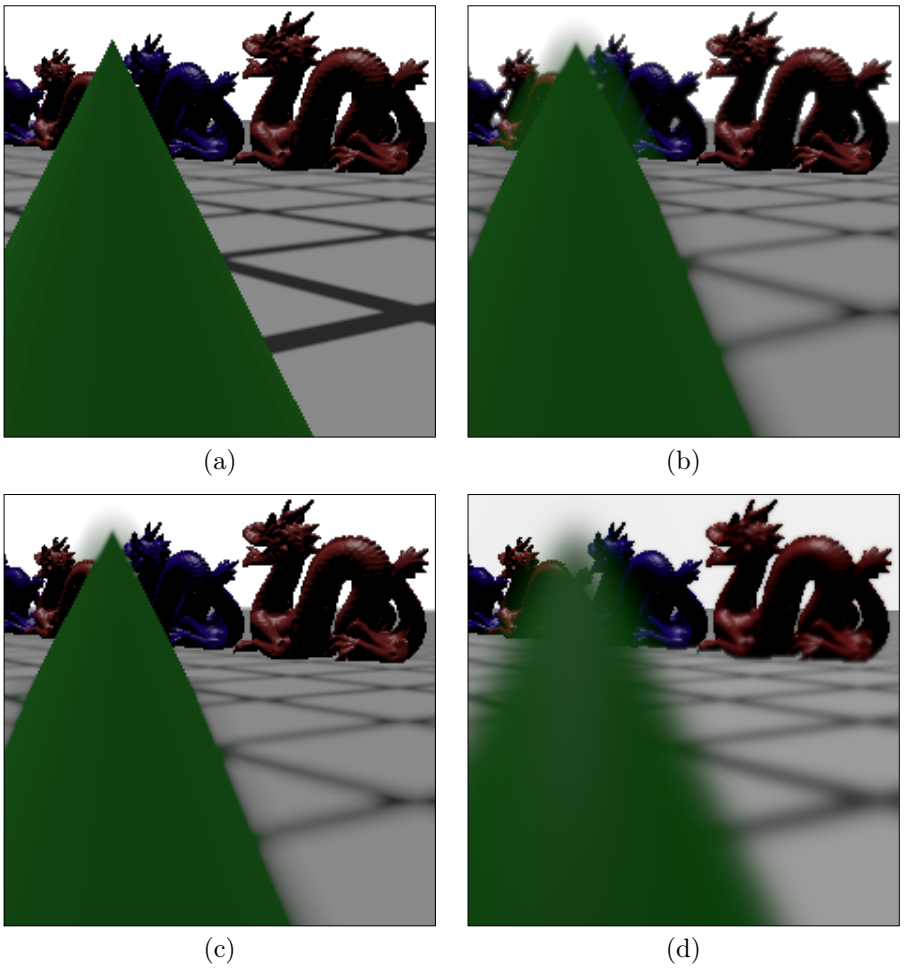


Figure 4.2: Images rendered with no DoF (a), with a post processing DoF method showing colour bleeding (b) and depth discontinuities (c), produced with Blender (from www.blender.org). As comparison, a DoF effect without partial occlusion problems (d), produced with the method described in chapter 5. In (b) the green cone bleeds its colour on the dragons in the background, while in (c) this behaviour is avoided by producing sharp borders. In (d) the green cone is smeared to semi-transparency, thus revealing the dragons in the background—strongly visible at the top of the cone. In single-layer post processing methods, such as (b) and (c), this transparency effect is not achievable.

Depth discontinuity artefacts

Depth discontinuity artefacts are artefacts strongly connected to intensity leakage and partial occlusion artefacts. This occurs if out-of-focus objects are in front of sharp objects. By preventing intensity leakage on sharp objects in the back, silhouettes of the out-of-focus foreground objects appear

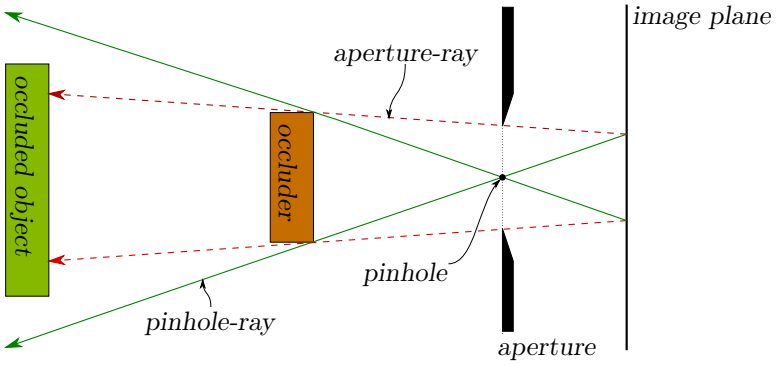


Figure 4.3: When a occluded object is completely hidden from rays traveling through a infinitely small aperture (pinhole-ray), it is possible that the occluded object can be reached by rays traveling through a finite aperture (aperture-ray). All rays start from the image plane located behind the aperture (adapted from [6]).

to be sharp. However, out-of-focus objects in the front should have soft borders and furthermore should be partly transparent so the background can bleed through. This transparency can be explained by partial occlusion. In figure 4.2 (c), a depth discontinuity artefact is shown where the green cone is out of focus but doesn't have smeared borders.

Partial Occlusion

One artefact, related to pixel bleeding (intensity leakage) and depth discontinuity artefacts, is the *partial occlusion* problem. In rasterised renderings, the pinhole-camera model is used which means the aperture is infinitely small. Depth of field is an effect which occurs at finitely aperture sizes (described in section 2.3). Rays hitting the image sensor can only pass a single point on the aperture if a pinhole-camera model is used. With a finite aperture, rays can scatter within the area of the aperture opening. Thus, making regions in the scene visible which are hidden in a pinhole model. The source of partial occlusion is illustrated, in figure 4.3. Ray tracing (see section 4.1) solves this problem. In image space methods, renderings are missing information, colour information, to fully resemble a finite aperture. Partial occlusion is noticeable at high aperture openings (where blurry objects near the camera are semi-transparent) resulting in partially visible background objects. Due to missing colour information of the occluded objects, the transparency effect cannot be achieved by simple filtering/blurring a single-layer rendering. Basically, this problem is apparent in all single-layered methods covered in section 4.2.2. Methods, revealing occluded objects, can produce DoF effects without partial occlusion artefacts. An example of this are layered methods as discussed in section 4.2.3.

Filtering artefacts

Since image space methods operate by filtering the scene rendering, artefacts caused by the blurring method can occur. Methods such as [46, 51, 68] use stochastic sampling, mipmapping, bilinear filtering, anisotropic filtering, or combinations of those filtering methods to increase the speed on graphics hardware. Using trilinear filtering (mipmapping) can cause blocky magnification artefacts (see section 3.4.2). Stochastic sampling, discussed in section 3.4.4, could result in jittering artefacts. Thus, DoF methods using fast inaccurate filtering methods might suffer artefacts related to blurring.

4.2.2 Single-layer methods

Single-layered DoF methods only operate on one single scene rendering and the depth buffer. Therefore, no re-rendering or modification of the scene is needed and interactive frame rates are possible. Since speed is important, such methods often use *gathering* and are used in interactive applications such as games.

Potmesil and Chakravarty (1981)

Potmesil and Chakravarty described the first method which artificially adds depth of field to computer renderings [62, 63]. It is a post processing approach using a linear filter to blur pixels according to their CoC. The method suffers from artefacts such as intensity leakage, depth discontinuity, and partial occlusion (see section 4.2.1). Despite the previously stated flaws, this technique is still the basis for modern DoF simulations and introduces equation 2.16: the formula for calculating the diameter of the CoC.

Rokita (1996)

Rokita proposed a method for virtual environments by repeating convolution with a 3×3 filter kernel, $\mathbf{H}$, of the form

$$\mathbf{H} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & b & 2 \\ 1 & 2 & 1 \end{bmatrix}, \tag{4.1}$$

where b is depending on the amount of blur [67]. Decreasing b increases the blur amount. Larger blurs can be achieved by repeating the convolution. The method is fast because it uses graphics hardware for filtering. However, it becomes slower with big blur sizes and suffers from the same artefacts as [62].

Fearing (1996)

Fearing introduces a DoF method based on [62] where the amount of preprocessing is dependent on the change of a pixel. Based on changes in depth, colour, and CoC, each pixel gets an importance value [28]. The more important pixels are rendered first, and the rendering process is stopped after a specified amount of pixels have been updated, a user interrupt, or a time interrupt occurred. Because of that fact, this method is well suited for animated scenes. Keeping pixels from the previous frame where no or only small changes occurred gains a speed increase (up to $\times 20$) compared to [62]. All artefacts from [62] are retained. Additionally, inconsistency artefacts occurring at too seldom updated pixels are introduced.

Mulder and van Liere (2000)

Mulder and van Liere present a technique making use of rendering hardware [56]. The authors propose two methods, one high quality and one faster (thus low quality), and an approach for using both methods simultaneously. The high resolution method is applied in the centre of the scene where important pixels lie (centre of attention) and the low resolution method is applied on the rest of the scene. For the high resolution method CoC diameters are discretised to pixel size and the border pixel of the different diameters are used. Rectangular polygons (their number is dependant on the amount of pixels in the CoC border) equal the size of the scene are drawn for each pixel and offset so that they align to the outermost border of the CoC border starting with the largest. Each texel of the polygon covers a pixel of the original scene. Only those pixels of each polygon that have a CoC larger than or equal to the current CoC border are rendered. This is done by using alpha coverage. Furthermore, with the alpha value, the intensity of a rendered border pixel is controlled. This process is repeated for the next smaller CoC diameter until the diameter is one. Thus, this method can be described as a *scattering* method because each pixel spreads its colour value based on a PSF (PSF is discussed in section 2.2).

The low quality approach is based on a *pyramid* method where the foreground is the level 0 and the background is composed of the levels in the pyramid. The number of levels is depending on the amount of blur.

The technique proposed by Mulder and van Liere produces good blurring results at the center of attention but produces blocky artefacts at positions where the lower quality method is applied. Furthermore, artefacts such as depth discontinuity and partial occlusion are not resolved with this technique.

Bertalmio et al. (2004)

Bertalmio et al. present a DoF simulation method using *heat diffusion* (as discussed in section 3.4.4) [9]. The conductivity is defined by

$$g(x,y) = \chi \left(\frac{|Z(x,y) - z_{\text{focus}}|}{Z(x,y)} \right)^2 . \tag{4.2}$$

Constant χ keeps g within the range $[0,1]$, $Z(x,y)$ is the depth-buffer value at (x,y), and z_{focus} is the focus distance.

The authors use a numerical iteration scheme to calculate the heat equation (equation 3.46). Backward differences[3] are used for the divergence and forward differences are used for the gradient, denoted by ∇^- and ∇^+

$$I_{m+1} = I_m + \alpha \nabla^- \cdot (g \cdot \nabla^+ I_m), \tag{4.3}$$

where $\alpha = 0.25$. Bigger maximum kernels are achieved by processing more iterations of diffusions (m) on the image.

Since the blur is nonuniform but depending on the CoC, intensity leaking is prevented. The method produces depth discontinuity artefacts (i.e., sharp borders) with out-of-focus foreground objects. Bertalmio et al. prevent the sharp borders of foreground objects by isotropic diffusion of the depth buffer at positions where the depth value is smaller than the focal plane. This method produces good results but suffers from partial occlusion problems such as most single-view post processing methods. At screen resolutions of 1024×768 pixels and $m = 20$ iterations, the approach is far from achieving real-time performance.

Kass et al. (2006)

Kass et al. propose a method based on heat diffusion with the purpose of producing high quality DoF simulations for off-line rendering [39]. 2D heat diffusion approaches are of complexity $\mathcal{O}(w^2)$ to produce a filter of width proportional to w. Therefore, the authors propose a separable approach where the heat diffuses along the u axis and along the v axis in a second step. Similar to separating a Gaussian filter, objectionable anisotropies are often avoided in the result.

The heat equation (equation 3.46) is solved by an backward Euler[4] approach resulting in a symmetric tridiagonal linear system. The conductivity

[3] The finite forward difference of a function f is defined as $\nabla^+ f(x) = f(x+1) - f(x)$, whereas the finite backward difference is defined by $\nabla^- f(x) = f(x) - f(x-1)$ (from http://mathworld.wolfram.com/FiniteDifference.html).

[4] The Euler backward method is an implicit method for solving an ordinary differential equation. In the case of a heat equation, for example, this means that a linear system must be solved at each time step. However, unlike the Euler forward method, the backward method is unconditionally stable and so allows large time steps to be taken (from http://mathworld.wolfram.com/EulerBackwardMethod.html).

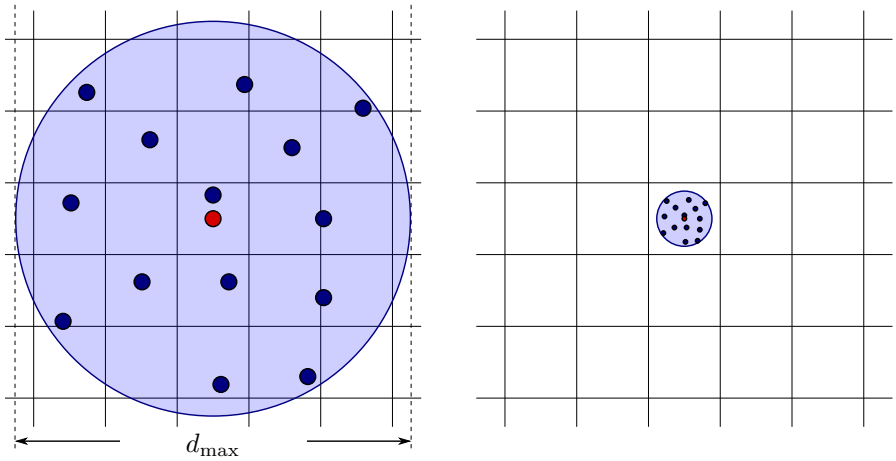


Figure 4.4: A point out of focus (blurriness = 1) and the corresponding Poisson disk size (a). If the point is in focus (blurriness = 0) the disk is scaled down to cover only one pixel (b) (adapted from [66]).

g is related to the CoC (equation 2.16). This linear system can be solved efficiently in constant time and on GPUs, making this method efficient.

The method prevents intensity leaking. However, sharp borders and artefacts at out-of-focus foreground objects appear, similarly to [9]. As a solution, the authors propose to use three layers: a foreground, midground, and a background layer. The use of more than one layer introduces additional rendering costs but solves the artefacts and also takes care of the partial occlusion problem. At a resolution of 1024×768 and one additional foreground layer, the frame rate is at 3 to 4 frames per second.

Riguer (2003)

Riguer describes two real-time DoF approaches showcasing it in an implementation for DirectX 9 [66]. The first method renders the scene with multiple render targets (MRT) where the colour is stored in one buffer (8-bit RGBA) and the depth and blurriness, calculated with equation 2.16, of a pixel is stored in a second buffer (2×16-bit). In the second rendering pass, the colour buffer is blurred with a 12-tap Poisson disk (as shown in figure 4.4). The Poisson disk is scaled according the amount of blurriness stored in the buffer. The maximum diameter $d_{\max}$ of the Poisson disk is predefined and limited by the number of taps in the disk (increasing the disk too much would result in noticeable artefacts). The scene texture is then sampled at the taps of the disk with bilinear filtering enabled. If a pixel is in focus, the Poisson disk is scaled down so that only one texel is sampled (see figure 4.4 (b)).

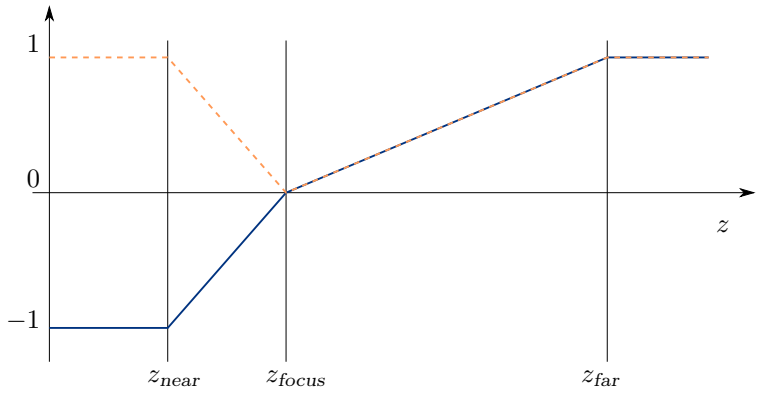


Figure 4.5: The relative depth (continuous line) and the mapping to the blur radius (dotted line) relative to d_{near}, d_{focus} and d_{far} in the range -1 to 1 and 0 to 1, respectively (adapted from [68]).

Each sample is averaged and contributes to the final colour of the processed pixel. This leads to intensity leakage which can be avoided by skipping taps in focus and in front of the centre pixel.

The second approach by Riguer uses Gaussian blur filtering. First, the scene is rendered into a frame buffer where the depth is stored in the alpha channel so MRT can be avoided. The scene texture is rendered to an off-screen buffer one fourth the size of the full resolution. The downsampled texture is blurred with a 25×25 Gaussian filter separated horizontally and vertically. In the final compositing step, the full resolution and low resolution scene textures are linearly interpolated based on a pixel's depth (stored in the alpha channel of the full-screen texture). Since the Gaussian blur is uniform, intensity leakage cannot be avoided with the second method proposed by Riguer.

Scheuermann and Tatarchuk (2004)

Scheuermann and Tatarchuk in [68] proposed a method which is basically an improvement and combination of the methods described in [66]. The scene is rendered into an off-screen buffer where the alpha channel stores the relative depth of the pixel. The depth is calculated relative to z_{near}, z_{focus}, and z_{far} as shown in figure 4.5 in the range -1 to 1 where z_{near}, z_{focus}, and z_{far} are set by an artist. Storing the depth in this way allows simple mapping to the blur radius d_{coc} by

$$d_{\text{coc}}(z) = d_{\max} \cdot |z|, \tag{4.4}$$

where $d_{\max}$ is the maximum CoC and z is the alpha value of the texel. This method avoids the usage of MRTs, but causes additional considerations when

dealing with transparent objects. Transparent objects must be rendered in two passes: the first one renders the colour and blends the object into the frame buffer and the second one writes the blur into the alpha channel. Before compositing the DoF effect, a blurred low-resolution version of the scene rendering is generated by downsampling it to a 1/16 of its original resolution with a 4×4 box filter. Blurring the final image is done by sampling a centre pixel and fetching its blur radius from the alpha channel. The Poisson disk is scaled according to this blur radius and used for sampling the high-resolution and low-resolution buffer (figures 4.4 (a) and (b)). Depending on the amount of blur, either the sharp high-resolution or the blurred low-resolution image contributes more to the final result. Intensity leakage can be minimized similarly to [66] by checking the depth and the sharpness of taps. Scheuermann and Tatarchuk provide the shader source code for their implementation. In [68] only 8 sample taps in the Poisson disk are used. Setting $d_{\max}$ too high results in aliasing artefacts due to low sampling rates. Partial occlusion artefacts cannot be avoided because the method does not use occluded pixels.

Hammon (2007)

Hammon compares DoF methods and proposes a method [34]. A gather method trying to resemble scattering is discussed first. The first method samples within the CoC (similar to [68]) but always uses the maximum circle of confusion. For every sample, the according CoC is calculated. Every sample is normalized according to its CoC, and the normalised sample is added to the center pixel. Thus, a scattering-similar approach is achieved at high computational costs. However, this method produces artefacts due to the fact that depth ordering is important and cannot be done at reasonable costs.

Due to these problems, the approach is skipped in favour of a visually appealing but physically inaccurate version. The foreground pixel's CoC are written in a texture where pixels in focus and behind the focal plane are set to zero. This texture is then blurred to avoid sharp borders for out-of-focus foreground objects (discontinuity artefacts). Additionally, the biggest CoC in a 2×2 neighbourhood (done with mipmaping) is used. The latter process produces CoC values without discontinuities in the foreground. For the background, the unmodified CoC is used. This approach is used for game engines, so the CoC is linearly approximated based on the depth of a pixel. Thus, artists have more control over the effect (similar to [68]). The final blurring is done by interpolating between three different blur radii speeded up by using a downsampled texture of the screen. The smallest and finest blur is done by 5 texture lookups averaging 17 pixels (using bilinear interpolation). According to the authors, this method uses an average of 9.6 texture lookups per pixel (which is fewer than [68]). This method avoids sharp borders for foreground

objects, which makes the partial occlusion problem less noticeable. However, partial occlusion cannot be resembled because the method is missing scene information. The fact that sharp objects can blur on unsharp background objects can be neglected because such artefacts are hardly noticeable.

Zhou et al. (2007)

In [74] a DoF method with the goal of producing a fast and accurate DoF effect is presented. Furthermore, one goal of the authors is to change as little as possible in an existing rendering pipeline so the depth buffer of OpenGL is used without further modifications.

The blurring of the rendering is done with a two-pass gather filter similar to separable Gaussian filters (section 3.4.1). First, the buffer is blurred vertically with a fixed width kernel. The weights of the filter are adaptive depending on the CoC (equation 2.16) of the center sample. Additionally, a simple PSF (the reciprocal of the square of the radius) and a factor to avoid intensity leakage are calculated and multiplied individually for each weight. Afterwards, the same filtering process is carried out horizontally.

The performance hit of this method on a scene with 420 000 triangles at a resolution of 1024×1024 and a kernel width of 9 pixels, are additional 25.2 milliseconds per frame on a NVIDIA[5] 6800 GPU. Increasing the width of the kernel results in greater performance hits (i.e., 70%) for a 13 pixels kernel.

The method avoids intensity leakage artefacts but suffers from partial occlusion artefacts and depth discontinuity artefacts. Since only small CoCs (13 pixels in [74]) are applicable, partial occlusion artefacts might not be strongly noticeable.

Lee et al. (2009)

A method designed for virtual reality environments running at high frame rates is discussed in [51]. The real-time performance is generated by using mipmaps for blurring. Although this has been used by other methods before, there are modifications to the standard mipmapping algorithm (section 3.4.4). Each mipmap level is generated with a 3×3 anisotropic Gaussian kernel to reduce intensity leakage artefacts. A sample tap of the Gaussian filter is rejected if the tap is sharper than the center tap (similar to [66]). Furthermore, to prevent blocky artefacts, known from mipmapping, the Gaussian filter taps are aligned in a circular as opposed to a rectangular grid. The mipmap levels can then be mapped to the standard deviation of a Gaussian

[5]NVIDIA is a leading manufacturer of graphic processing units (http://www.nvidia.com).

filter which furthermore are mapped to the CoC by

$$\sigma = \frac{d_{\text{coc}}}{2}. \tag{4.5}$$

The usage of the anisotropic mipmaps reduces colour leaking artefacts but produces sharp borders (depth discontinuity artefacts). Lee et al. solve this by smoothing foreground boundaries in the buffer storing σ which is a similar approach as the one used in [34].

The two DoF artefacts (i.e., intensity leakage and depth discontinuity artefacts) are thus efficiently reduced. Partial occlusion problems are hardly noticeable at smaller blur radii, because they are hidden by the fact that unsharp foreground objects are smeared over sharp background objects. However, this is not an optically correct solution for partial occlusion and fails at high blur radii where the foreground object should be semitransparent.

The performance of this method is dependant on the scene and the mean CoC size. Yet, according to the figures in [51] it is, on average, approximately 20% slower than [68].

Kosloff (2010)

The novel contribution of the thesis [40] is the comparison of spreading and gathering when performing those operations as matrix multiplications. The following filtering methods are presented: rectangle spreading, perimeter method, polynomial spreading, spreading with pyramid levels, and a tensor method. However, only one of these methods is used in an DoF implementation. For an AMD[6] DoF demo, the filtering method called polynomial spreading is used with an implementation tailored to DirectX 11 graphics hardware.

Polynomial spreading uses a similar method to the one proposed by Heckbert in [35] (section 3.4.4). This method filters an image with the mth derivative and transforms it into the final output by taking the mth integral. The advantage of this is that for example, a quadratic impulse response has a sparse third derivative thus producing only a few texture writes.

Although few texture writes are an indication for speed-ups, this is not the case on pre DirectX 11 graphics cards since the polynomial spreading is a spreading method (which are slow on GPUs). In [40], the author proposes an implementation using scatter and atomic operations of the DirectX 11 standard. To take full advantage of the parallelism of modern GPUs the integration step has to be carried out in domains requiring one additional fixing path.

Kosloff's method reaches over 30 FPS on an ATI Radeon HD58xx series video card and, since it is a spreading method, avoids artefacts such as

[6] AMD is a manufacturer of central processing units and ATI graphic processing units (http://www.amd.com).

intensity leakage or depth discontinuity effects. Thus, partial occlusion is hardly noticeable. However, if the scene input is only one single rendering, not containing layers or other information, the partial occlusion problem cannot be handled correctly.

4.2.3 Multi-layer methods

In multi-layered methods a scene is either decomposed or rendered into layers, and therefore these methods allow new ways of filtering and resolving artefacts. Producing and composing layers introduces additional rendering effort. Thus, layered methods usually operate at lower frame rates than single-layer methods. While all single-layer methods suffer the partial occlusion problem to some degree, multi-layered methods can resolve this issue.

Barsky et al. (2005)

In [7, 8], the authors investigate artefacts produced by layering methods, such as occlusion and discretisation problems. While the partial occlusion problem has already been discussed, the discretisation problem only appears in layered methods.

A rendering of a scene is used and split into layers. Each layer contains pixels with depth values within a certain depth range. Thus, layers can be blurred uniformly and efficiently with a gathering method. However, the discretisation of a pixel into only one layer introduces artefacts if layers are blurred. The border of an object in one layer gets blurred and therefore smeared out. When this smeared-out border is blended on the other layers, this smeared region appears as a ringing artefact due to the reduced opacity.

In [8], those discretisation artefacts are reduced by extending objects at layer boundaries. One method to do so is by edge detecting the depth buffer or by a first degree difference map. With these methods, object boundaries within a layer are determined and the layer can be extended by neighbouring pixels. Because of those extended sub-images, discretisation artefacts are not visible.

However, since there is no other scene information than one single rendering, high blur radii leading to semitransparent foreground objects cannot be simulated. Thus, the partial occlusion problem cannot be resolved with this method.

Kraus and Strengert (2007)

Kraus and Strengert's method in [46] follows a similar approach as [8] which is the decomposition of a single scene rendering into multiple layers and blurring them uniformly. Layer boundaries z_i for splitting the scene rendering

into sub-images are calculated by

$$z_i = \begin{cases} \frac{z_{\text{focus}}}{1+d_{\text{pix}}(i)/d_{\max}} & \text{for } i < 0, \\ z_{\text{focus}} & \text{for } i = 0, \\ \frac{z_{\text{focus}}}{1-d_{\text{pix}}(i)/d_{\max}} & \text{for } i > 0, \end{cases} \tag{4.6}$$

where $i \in \mathbb{Z}$ and $d_{\max}$ is defined by

$$d_{\max} = \frac{w_{\text{pix}}}{w} \cdot \frac{f}{N}. \tag{4.7}$$

The variables w_{pix} and w are the width of the image and the image sensor in pixel and meters. N and f are the f-stop and the focal length respectively. The diameter $d_{\text{pix}}(i)$ is dependant on the blur-method chosen. Basically, various methods from section 3.4.4 could be applied. In [46] it is approximated by

$$d_{\text{pix}}(i) \approx \begin{cases} 0 & \text{for } i = 0, \\ 0.85 \times 2^{|i|-1} & \text{for } i \neq 0. \end{cases} \tag{4.8}$$

The boundaries in a plot are shown in figure 6.7 (b).

With those layer boundaries, the scene rendering gets decomposed into layers. In the next step foreground pixels in each layer, which are in front of the particular layer boundary z_i, are culled away. This results in undefined areas in each layer. Therefore, these culled pixels are disoccluded which means they are filled by interpolated colour and depth values. This disocclusion is done with the pyramid method described in [45, 72]. To avoid discretisation artefacts, the layers do not contain any undefined colour values. After disocclusion, the layers are matted with a matting function. The matting function is discussed in detail in section 5.2 because this thesis' method reuses some parts of that function. The purpose of the matting function is to cull away pixels in the foreground and background which do not belong to a layer. For an exemplary layer L_i the core depth range is z_{i-1} to z_i. Additionally, the matting function of [46] introduces a smooth transition to avoid hard cuts and thus resulting in a extended depth range of z_{i-2} to $z_i + 1$ for layer L_i. Since every layer contains pixels of similar depth, thus similar CoC, each layer can be blurred uniformly, done by a pyramid method from [45, 72]. Afterwards, the blurred layers are blended from back to front resulting in a DoF effect without discretisation artefacts. Artefacts at the boundaries of the scene, caused by pyramid blurring, are avoided by the fact that the sub-images are in a buffer of a larger size than the original rendering.

The authors Kraus and Strengert compare their method to renderings produced with the ray tracer pbrt. According to the authors, the layers are slightly over blurred due to the usage of the matting function. Despite that, results look appealing. The results are not optically accurate, since scene

information is interpolated. This might become apparent at high blur radii where foreground objects are semi-transparent and should reveal in-focus objects behind them.

The performance of this method depends on the amount of layers used which is dependant on the maximum CoC, $d_{\max}$. On a NVIDIA Geforce 7900 GTX GPU with a resolution of 1024×1024 pixels, 12 sub-images result in 14.2 FPS and 6 layers in 29 FPS.

Kosloff and Barsky (2007)

In [41], the authors propose a generalised depth of field method. The basic idea is to use DoF to anticipate objects in a scene without optical constraints. For example, the in-focus area could be a region/volume shaped like a cross or two objects at completely different depths are in focus while the rest of the scene is out of focus.

A scene is rendered in layers where each layer contains one object of the scene. A layer has a position map storing x, y and z coordinates of the pixels. Blur values are calculated based on the position map for each pixel in the layers. Based on these blur values, the layers are blurred with a heat diffusion (as discussed in section 3.4.4). The borders of an object act as bounds for the heat diffusion, and thus preventing objects from being transparent at in-object regions. After blurring, the layers are composed from back to front resulting in a generalised depth of field effect.

Because of the fact that anisotropic filtering is used, objects spanning over a high depth range do not have to be split. Partial occlusion and discretisation artefacts are handled well by the method because occluded scene information is available in layers and objects are not split across layers. However, the method cannot handle complex objects occluding itself which is a problem, especially with complex scenes. Furthermore, the rendering of each scene object separately and the usage of heat diffusion results in frame rates far from interactive rates.

Lee et al. (2008)

In [49], a method based on splatting which resolves the partial occlusion method is presented. Hidden scene information is rendered with a technique called depth peeling, as discussed in section 5.1 because it is used in this work's method. Additionally, the peeling method is extended by edge-detection on the depth buffer so that only pixels relevant for partial occlusion are stored.

The scene rendering and the hidden scene renderings are splatted onto three pixel layers, for each incoming image, via MRT. The PSF used for spreading the colour of a pixel is stored in a LUT. While spreading, an incoming source pixel is assigned to one of the three layers (near, same, and

farther layer) with respect to the destinations pixel's depth. Thus, avoiding the sorting problem known from spreading DoF methods. The generated layers containing the spread values are blended from back to front. Afterwards, the compositing has to be normalised to avoid normalisation artefacts.

This method produces appealing results with respect to partial occlusion. However, the splatting method is computationally expensive and therefore the method runs at non-interactive frame rates. The authors propose an accelerated method where the resolution of the sprite, for splatting, is reduced. An NVIDIA Geforce 9800GX2 GPU, a resolution of 1024×768 pixels, a mean COC size of 32 pixels, and a scene with approximately 220 000 triangles produced a frame rate of 1 FPS without acceleration and 31 FPS with the accelerated method.

Lee et al. (2009)

In [50], a scene is rendered into K layers. Each layer consists of renderings of the scene with the near and far clipping plane set to the layer boundaries. Thereby producing hard cuts between layers (in practice the layer overlap) and also producing important scene information needed for partial occlusion.

The layers are split uniformly with respect to the change of the CoC, Δ_d. First the maximum circle of confusion $d_{\max}$ (which is at the near clipping plane z_{near}, if the focal plane is set to infinity $z_{\text{focus}} \to \infty$) is calculated. With these constraints $d_{\max}$ can be approximated by the equation 2.16 resulting in

$$d_{\max} = d(z_{\text{near}}, f, b, z_{\text{focus}} \to \infty) \approx \left| -\frac{f^2}{z_{\text{near}} \cdot N} \right|. \tag{4.9}$$

Then the change of CoC for each layer is defined by

$$\Delta_d = \frac{d_{\max}}{K}. \tag{4.10}$$

The layer index k for a pixel/fragment q with the depth z_{q} is given by

$$k = \left\lfloor \frac{af}{\Delta_d \cdot (z_{\text{focus}} - f)} \cdot \left(\frac{1}{z_{\text{near}}} - \frac{1}{z_{\mathsf{q}}} \right) \right\rfloor, \tag{4.11}$$

where z_{q} lies within the near and far clipping planes ($z_{\text{near}} < z_{\mathsf{q}} < z_{\text{far}}$). A plot of the boundaries with respect to depth and the CoC is shown in figure 6.7 (c).

After rendering a layer, its depth buffer is used as a height-map for ray traversal. Instead of using image manipulation methods (spreading, gathering or heat diffusion), the depth of field effect is generated by a simple form of ray tracing. The ray traversal method is called *cone tracing* and artefacts are reduced by jittered sampling (jittering is explained in section 3.4.4). Thus,

discretisation or normalization artefacts known from other layered methods are avoided. Partial occlusion is handled correctly since occluded information of the scene is used.

The rendering of layers is produced with DirectX's *array textures* allowing individual depth buffers per layer. The performance of the method is dependant on the number of layers K and the number of views/rays V. With $V = 64$ (which is sufficient according to the authors) and $K = 16$ a scene with 300 000 triangles runs at a frame rate of 24 FPS.

Kosloff et al. (2009)

In [43] a DoF method based on spreading is introduced. The spreading is based on a similar approach as discussed in [35]. A derivative of a rectangular filtering kernel, the method is called rectangular spreading, is used on a summed area table. Thus, only four texture writes are necessary for spreading one pixel with a filter kernel of any size. Normalization of the filter is done with an additional fourth channel initially containing 1. Point primitive generation features of DirectX 10 are used for an efficient implementation.

Furthermore, the authors propose a second method to spread filters of any shape by writing values for each scanline. Thus, the performance is depending on the perimeter of the chosen PSF.

A hybrid of the two proposed spreading methods is discussed: low contrast areas are blurred with rectangular spreading, and areas of high contrast (where simple PSFs are more noticable) are blurred with the more complex spreading method.

At a resolution of 800×600, pixels a framerate of 45 FPS can be achieved with rectangular spreading on an ATI HD4870 GPU.

Since [43] uses layers, partial occlusion can be solved. The layer decomposition is kept simple (similarly to [41]) and therefore objects cannot occlude themselves. Thus, discretisation artefacts are not an issue. The authors claim that their method is also applicable for simple scenes without layers, but this would result in partial occlusion problems.

Lee et al. (2010)

In [52], an extension/improvement to [50] is presented. Similar to [50], the scene is rendered into layers and the DoF effect is generated by ray traversal. The method in [52] differs by using depth peeling for the layer creation and additionally simulating other lens effects, such as tilt-shifted lenses, chromatic aberration (section 2.1.4), and others. The depth peeling (discussed in section 5.1) is accelerated by ignoring pixels that cannot be hit by any lens ray (due to occlusion). Thus, the amount of depth peeling layers can be reduced. This results in speedups during the composition.

The generated layers are composed via ray traversal. Rays are modified

according to the simulated lens effects (lens effects are discussed in section 2.1).

Because of the optimized depth peeling, the authors claim that 4 layers are sufficient in most cases. The performance hit of the DoF effect on a scene with 400 000 triangles, 100 lens rays, 4 layers, and a resolution of 800×600 on an NVIDIA Geforce 285GTX GPU is 24 milliseconds.

Partial occlusion is handled correctly. Since a form of ray traversing is used, discretisation artefacts are not a problem. However, artefacts slightly occurring are noise artefacts, caused by too few lens rays.

4.3 Summary

Methods for generating DoF effects have been discussed in the previous sections. The conclusion of this chapter is:

- Either the method produces physically accurate DoF effects, such as ray tracing or the accumulation buffer method (section 4.1), but generates high rendering costs.
- Or, a rendering approach tries to approximate DoF by altering a already rendered scene (section 4.2) at reasonable rendering costs.

Although the first mentioned methods produce accurate DoF renderings, the latter are of interest in this thesis.

In single-layer methods, altering a single image is done by filtering with spreading or gathering filters (section 4.2.2). For renderings at interactive frame rates, fast filtering methods are used (section 3.4). The blurring shape and size of pixels are determined by the optical constraints of cameras (chapter 2). The model of a simple thin lens (section 2.1.2) and the lens' CoC (section 2.3) is sufficient for DoF approximations. Reasonable results can be achieved with single-layer filtering methods; especially if the filtering method is spreading or heat diffusion, e.g., [39, 40]. Nevertheless, single-layer methods cannot handle partial occlusion correctly.

If a solid approximation of partial occlusion is the goal, then multi-layer DoF methods have to be used. Methods, successfully handling partial occlusion, use more than one input image for the DoF approximation (section 4.2.3). Furthermore, with the layered representation it is possible to uniformly blur a layer. Thus, partial occlusion can be handled correctly, but layer discretisation artefacts might occur. Recent multi-layer methods (i.e., [50, 52]) try to avoid such artefacts by ray traversing.

In the following chapter, a multi-layer DoF method is proposed reusing some of the ideas discussed in this chapter.

Chapter 5

Proposed method

A multi-layer approach, similar to methods discussed in section 4.2.3, is presented in this work. The rendered scene is decomposed into layers where each layer contains pixels of a certain depth range. This decomposition approach is similar to layered methods like [8, 41, 43, 46]. However, one main difference to the prior mentioned methods is that there are two scene buffers where the second buffer contains a rendering where only occluded pixels are stored (section 5.1). Thus, partial occlusion artefacts are avoided. Additionally, the matting function from [46] prevents discretisation artefacts (section 5.2). The decomposed layers are blurred according to their CoCs (section 5.3) and blended back to front for a final result (section 5.4). Furthermore, a method for combining, thus efficiently computing, the blur and the layer composition is proposed (section 5.5).

With the occluded scene information, the discussed approach avoids partial occlusion artefacts. Thus, high out-of-focus objects can be blurred to transparency and reveal the background. While [51] also uses depth peeling to avoid partial occlusion issues, the authors use spreading for blurring the scene. This work's method uses uniform filtering for each layer and so gathering filters are applied.

The scene is rendered into layers in [50]. Blurring the scene, in [50], is done by intersection testing within a layer thereby avoiding the discretisation problems but introducing noise. The method proposed in this thesis is free of such noise.

The depth of field method discussed in this thesis can be structured into the following steps:

1. Render the scene into a buffer I_0 containing the colour information—red, green, blue, and alpha—and a depth buffer Z_0.
2. With the depth buffer from the previous rendering step Z_0, this method will depth peel the scene and store the colour and the depth into buffers I_1 and Z_1 respectively (section 5.1).
3. Decompose I_0 and I_1 into K layers L_0 to L_{K-1} with a matting function

based on the pixel's depth values in Z_0 and Z_1 (section 5.2).
4. Filter each layer L_k based on the appropriate CoC (section 5.3).
5. Compose the processed layers L'_0 to L'_{K-1} from back to front (section 5.4).

Figure 5.1 outlines the above described algorithm.

5.1 Rendering partly occluded objects

For a post-process depth-of-field method to fully resemble the partial occlusion effect (described in section 4.2.1), additional scene and depth buffers are needed. One way of resolving this issue is to render the scene in K layers by setting the near and far clipping planes (z_{near} and z_{far}, discussed in section 3.1) of the camera as it has been done in [8, 50, 69]. Although rendering into layers would simplify the decomposing step (but not completely remove it, because of matting), a technique called *depth peeling* is used to reduce the amount of scene renderings. Rendering a scene a number of times, even with modified near and far clipping planes, is more expensive than post processing (especially if the scene is complex). *Depth peeling* got introduced in [27] and its basic idea is to use a similar technique as shadow mapping.[1]

5.1.1 Depth peeling

For depth peeling, first a 3D scene is rendered into a buffer storing the colour I_0 and the depth Z_0 of a rendering, shown in figures 5.2 (a) and (b), respectively. Then the scene is rendered a second time into new buffers I_1 and Z_1 while projecting the depth buffer Z_0 onto the scene. For each fragment p with the coordinates $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p}, z_\mathsf{p})$ and the projected coordinates $\mathsf{p}' = (x_{\mathsf{p}'}, y_{\mathsf{p}'}, z_{\mathsf{p}'})$, a depth test with the projected depth buffer is carried out in the fragment shader. The fragment p gets rejected if its depth z_p has the same or smaller depth than the previously rendered fragment stored in I_0 and Z_0. Resulting in

$$I_1(x_{\mathsf{p}'}, y_{\mathsf{p}'}) \leftarrow \begin{cases} \text{colour}(\mathsf{p}) & \text{for} \quad Z_0(x_{\mathsf{p}'}, y_{\mathsf{p}'}) < z_\mathsf{p}, \\ I_1(x_{\mathsf{p}'}, y_{\mathsf{p}'}) & \text{otherwise}, \end{cases} \tag{5.1}$$

and

$$Z_1(x_{\mathsf{p}'}, y_{\mathsf{p}'}) \leftarrow \begin{cases} z_\mathsf{p} & \text{for} \quad Z_0(x_{\mathsf{p}'}, y_{\mathsf{p}'}) < z_\mathsf{p}, \\ Z_1(x_{\mathsf{p}'}, y_{\mathsf{p}'}) & \text{otherwise}, \end{cases} \tag{5.2}$$

meaning that only previously occluded fragments are stored in I_1 and Z_1 (shown in figures 5.2 (c) and (d)). If a fragment is rejected it is "peeled away"

[1] Shadow mapping is a technique introduced in [48] and widely used for simulating shadows in 3D environments.

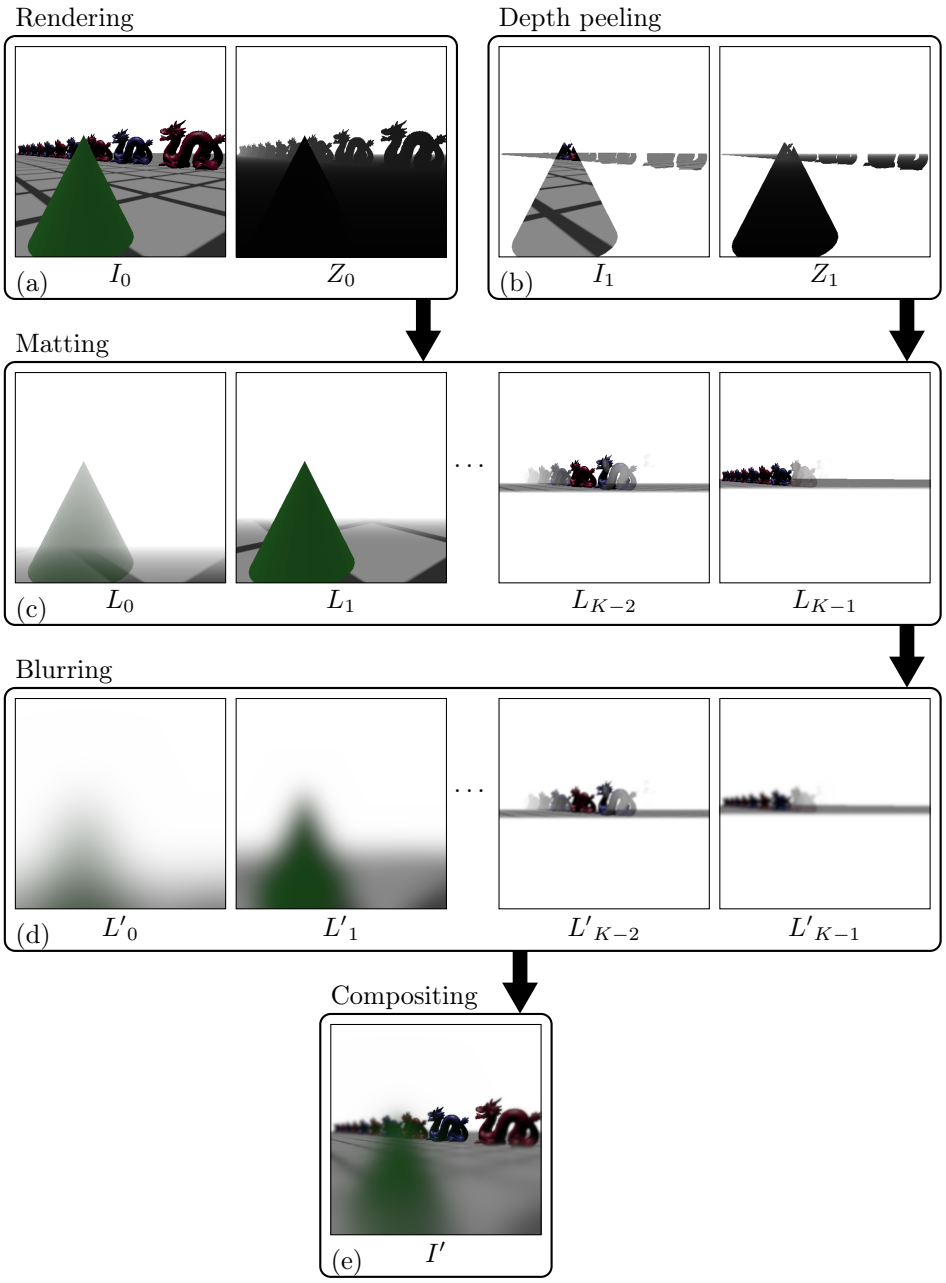


Figure 5.1: A overview of the proposed method in this work: The scene and the depth are rendered into buffers I_0 and Z_0 (a). Via depth peeling, occluded scene information is revealed and stored in I_1 and the peeled depth stored in Z_1 (b). I_0 and I_1 are decomposed into K layers L_0 to L_{K-1} by matting (c). The decomposed layers get blurred based on the depth in each layer resulting in L'_0 to L'_{K-1} (d). Finally, the processed layers are composed to a final image I', (e).

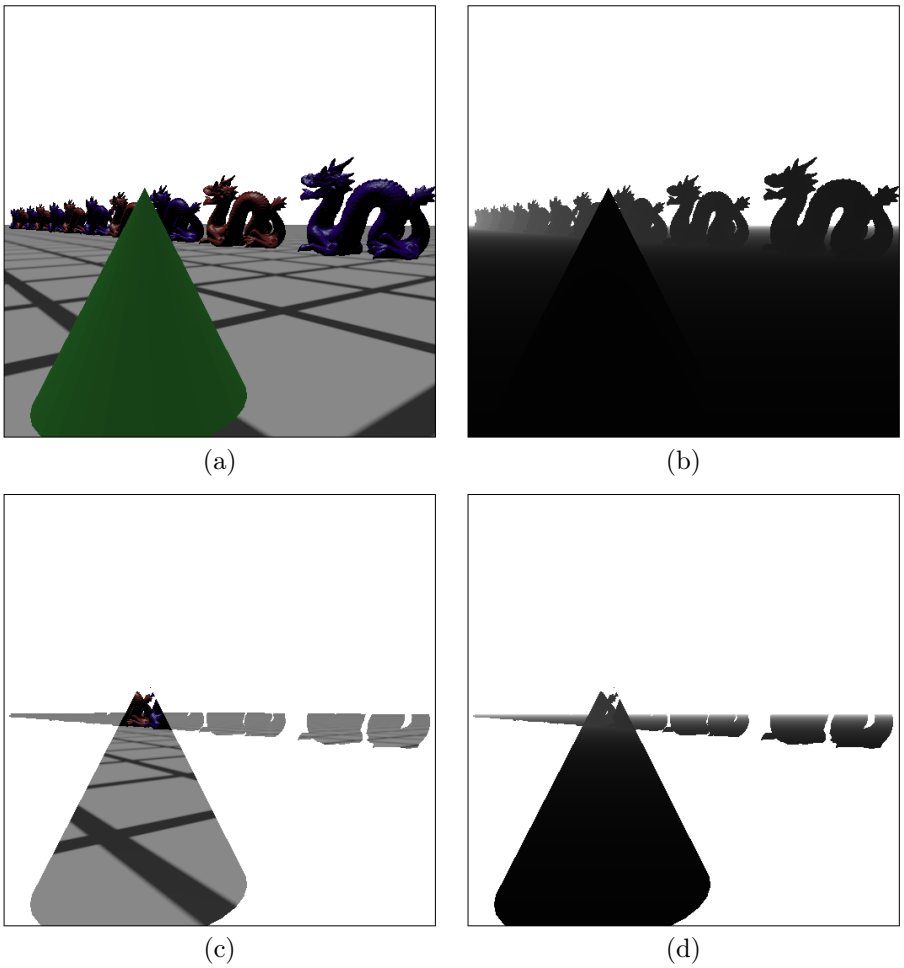


Figure 5.2: The rendering of a scene I_0 (a) and its depth buffer Z_0 (b). A second rendering of the scene produced with *depth peeling* resulting in a peeled scene I_1 (c) and a peeled depth buffer Z_1 (d).

revealing objects behind the first layer. Depth peeling could be executed recursively (always use Z_{k-1} to render layer I_k and Z_k) to disclose further layers. A stopping condition might be a certain amount of layers or the number of fragments not rejected in the current peeling pass (0 for disclosing all fragments in a scene). However, the proposed implementation in this paper only uses one hidden layer. Every peeling step renders the whole scene and therefore more layers result in additional costs.

5.1.2 Further considerations

A simple depth test, such as in equation 5.1, might fail because of precision variance caused by the limited precision depth buffer. Therefore, a minimum

distance ϵ between two peeled layers gets introduced. The *for* condition of equations 5.1 and 5.2 is then redefined to

$$Z_0(x_{\mathsf{p}'}, y_{\mathsf{p}'}) + \epsilon < z_{\mathsf{p}}. \tag{5.3}$$

Similarly to shadow mapping, choosing an ϵ for producing reasonable results is scene depending. If a pixel in the first rendering $I_0(x_{\mathsf{p}'}, y_{\mathsf{p}'})$ and the peeled sub-image $I_1(x_{\mathsf{p}'}, y_{\mathsf{p}'})$ are both decomposed into the same layer L_k and the pixels' depths are similar $Z_0(x_{\mathsf{p}'}, y_{\mathsf{p}'}) \approx Z_1(x_{\mathsf{p}'}, y_{\mathsf{p}'})$, then the colour from I_1 in L_k will be overwritten (see section 5.2). Therefore, an approach would be to set ϵ based on the layer boundaries discussed in section 5.2 to avoid overwriting peeled pixels.

In [49], depth peeling is used with some additional constraints: Edge detection in the depth buffer is used only to reveal occluded fragments at borders of depth discontinuities. Additionally, fragments behind the focal plane ($z_{\mathsf{q}} > z_{\text{focus}}$, where z_{focus} is the focal plane) are discarded because the partial occlusion problem is hardly noticeable at such areas. Such optimisations might also be an option for the method described in this work when performance improvements would be needed.

5.2 Scene decomposition

The rendering I_0 and the peeled scene I_1 are decomposed into K layers $L_0, L_1, \ldots, L_{(K-1)}$. Each layer L_k contains pixels (consisting of red, green, blue, and alpha channels) from the buffers I_0 and I_1. Which pixel is composed into a layer is given by a matting function $\omega(z)$, where z is the pixel's depth. Therefore, a layer L_k is defined by

$$L_k = \Big(I_0 \cdot \omega_k(Z_0)\Big) \oplus \Big(I_1 \cdot \omega_k(Z_1)\Big), \tag{5.4}$$

where $\omega_k(z)$ denotes the matting function for the layer L_k. Thus, all pixels decomposed in a layer L_k have a similar CoC necessary for uniformly blurring (see section 5.3).

5.2.1 Matting function (ω)

The matting function is taken from [46] and defined as

$$\omega_k(z) = \begin{cases} \frac{z - z_{k-2}}{z_{k-1} - z_{k-2}} & \text{for} \quad z_{k-2} < z < z_{k-1}, \\ 1 & \text{for} \quad z_{k-1} \leq z \leq z_k, \\ \frac{z_k - z}{z_k - z_{k+1}} & \text{for} \quad z_k < z < z_{k+1}, \\ 0 & \text{otherwise}, \end{cases} \tag{5.5}$$

where z_{k-2} to z_{k+1} defines anchor points for the layer matting. A plot of the function is shown in figure 5.3 for exemplary anchor points. Since the

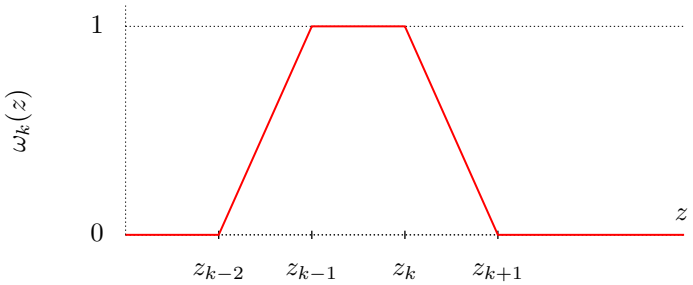


Figure 5.3: A plot of the matting function $\omega_k(z)$ for the layer L_k with exemplary depth coordinates z_{k-2} to z_{k+1}.

rendered scene and the peeled scene are weighted with $\omega(z)$, the function might be also called a weighting function. The matting function is defined continuously for smooth transitions of pixels across three layers. Since the function is not normalised, it will contribute a summed weight of 2. There are K anchor points. Because of this the matting functions ω_0, ω_1, and ω_{K-1} are ill-defined with respect to equation 5.5. Therefore, the matting functions for those layers are:

$$\omega_0(z) = \begin{cases} 1 & \text{for} \quad z \leq z_0, \\ \frac{z_0 - z}{z_0 - z_1} & \text{for} \quad z_0 < z < z_1, \\ 0 & \text{otherwise}, \end{cases} \tag{5.6}$$

$$\omega_1(z) = \begin{cases} 1 & \text{for} \quad z \leq z_1, \\ \frac{z_1 - z}{z_1 - z_2} & \text{for} \quad z_1 < z < z_2, \\ 0 & \text{otherwise}, \end{cases} \tag{5.7}$$

$$\omega_{K-1}(z) = \begin{cases} \frac{z - z_{K-3}}{z_{K-2} - z_{K-3}} & \text{for} \quad z_{K-3} < z < z_{K-2}, \\ 1 & \text{for} \quad z_{K-2} \leq z, \\ 0 & \text{otherwise}. \end{cases} \tag{5.8}$$

After decomposition, a pixel from I_0 or I_1 has no longer the unique depth Z_0 or Z_1, respectively, but for example the depth range z_{k-1} to z_k for the layer L_k.

Other matting functions

The weighting function ω_k might produce artefacts in some cases (further explained in section 6.2). Therefore, this section proposes two alternative matting functions $\dot{\omega}_k$ and $\ddot{\omega}_k$ for the use in equation 5.4. Different to equation 5.5, the first function prevents all pixels with a depth greater than z_k

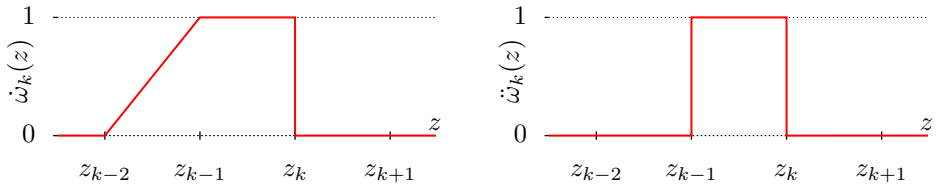


Figure 5.4: Alternative matting functions $\dot{\omega}_k$ (a) and $\ddot{\omega}_k$ (b) for the layer L_k with exemplary depth coordinates z_{k-2} to z_{k+1}.

from contributing any colour to layer L_k:

$$\dot{\omega}_k(z) = \begin{cases} \frac{z - z_{k-2}}{z_{k-1} - z_{k-2}} & \text{for} \quad z_{k-2} < z < z_{k-1}, \\ 1 & \text{for} \quad z_{k-1} \leq z \leq z_k, \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

Therefore, there is a hard cut boundary at the back of a layer which produces layering artefacts as discussed in [8] but hinders the peeled background to bleed through (section 6.2). Another matting function $\ddot{\omega}_k$ produces hard layer cuts in the back and the front layer borders. Only pixels in the range z_{k-1} to z_k are excepted in layer L_k:

$$\ddot{\omega}_k(z) = \begin{cases} 1 & \text{for} \quad z_{k-1} \leq z \leq z_k, \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

Usage scenarios for the other matting functions are discussed in section 6.2. Figures 5.4 (a) and (b) show plots of the matting functions.

5.2.2 Layer boundaries

In layered DoF methods, there are various approaches for spacing layers (see section 4.2.3). In the approach discussed, K layers $(L_0, L_1, \ldots, L_{K-1})$ are generated by matting with a matting function. The matting function relies on K anchor points $(z_0, z_1, \ldots, z_{k_{\text{focus}}}, \ldots, z_{K-1})$ where k_{focus} is the index of the layer in focus. In [50], the layers are split with respect to the change in the CoC. A maximum circle of confusion $d_{\max}$ is used and split by the number of layers. In [46], the boundaries are spaced accordingly to the chosen blurring method. This work's method follows a similar approach where the formula for calculating the CoC (equation 2.16) is rearranged to calculate a depth z based on a given CoC d. Since d_{coc} is non-injective. There are two possible results

$$d_{\text{coc}}^{-1}(d) = \big(D_1(d), D_2(d)\big) \tag{5.11}$$

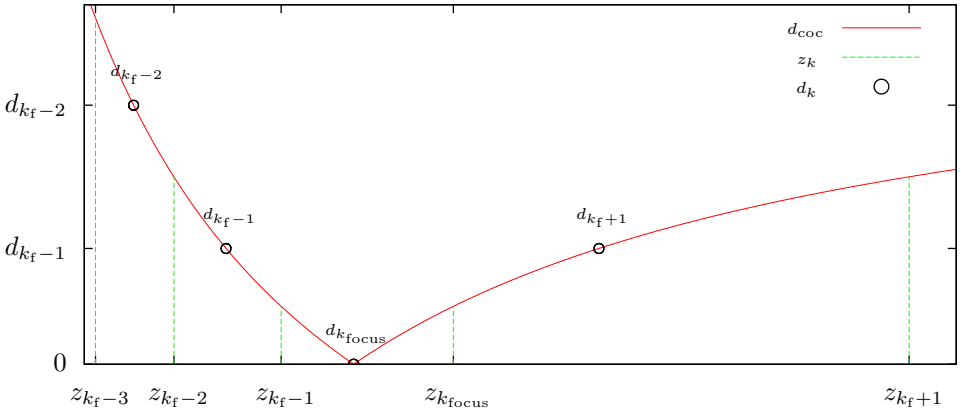


Figure 5.5: A plot of d_{coc} with exemplary anchor points $z_{k_{\mathrm{focus}}-3}$ to $z_{k_{\mathrm{focus}}+1}$ marking the layer bounds (equation 5.14). The Layer L_k has a CoC of d_k, and the CoCs are marked as $d_{k_{\mathrm{focus}}-2}$ to $d_{k_{\mathrm{focus}}+1}$. Note that k_{focus} has been shortened to k_{f} for this plot.

with

$$\big(D_1(d), D_2(d)\big) = \left(\frac{z_{\mathrm{focus}} \cdot f^2}{f^2 + d \cdot N \cdot (z_{\mathrm{focus}} - f)}, \frac{z_{\mathrm{focus}} \cdot f^2}{f^2 - d \cdot N \cdot (z_{\mathrm{focus}} - f)}\right). \tag{5.12}$$

Note that $D_2(d)$, $d \in \mathbb{R}^+$ is only applicable as long as

$$d < \frac{f^2}{N \cdot (z_{\mathrm{focus}} - f)}, \tag{5.13}$$

otherwise $D_2(d)$ will be negative and not applicable as an anchor point. The anchor point furthest away, z_{K-1}, is bound by this constraint.

With the use of the matting functions, points in the depth range z_{k-1} to z_k contribute their unbiased colour to the layer L_k. Depending on the weighting functions ω, $\dot{\omega}$, and $\ddot{\omega}$ the depth range might extend to: z_{k-2} to z_{k+1}, z_{k-1} to z_{k+1}, and z_{k-1} to z_k. Therefore, the anchor point z_k should be placed in the middle of the layers L_k and L_{k+1} as

$$z_k = \begin{cases} D_1\left(\frac{d_k + d_{k+1}}{2}\right) & \text{for} \quad k < k_{\mathrm{focus}}, \\ D_2\left(\frac{d_k + d_{k+1}}{2}\right) & \text{for} \quad k \geq k_{\mathrm{focus}}, \end{cases} \tag{5.14}$$

where d_k and d_{k+1} are the CoCs of the layers L_k and L_{k+1} respectively. A plot of the CoC with exemplary anchor points is shown in figure 5.5.

5.3 Blurring

The layer L_k, containing pixels of a certain depth range $(z_k - z_{k-1})$, is blurred uniformly by a filter kernel. The layer gets blurred with a convolution method denoted by

$$L'_k = L_k * H_k, \tag{5.15}$$

where H_k is a filter kernel based on d_k (the CoC of the layer L_k). Basically, various filtering methods discussed in section 3.4 can be used. Note that the blurring is done with associated colour values (section 3.3.3). This thesis' method uses Gaussian filters for blurring the layers. An optimisation for this method uses recursive filters (see section 5.5). Gaussian filters are used in other post-processing DoF implementations (section 4.2).

The anchor points, z_0 to z_{K-1}, for the layer matting are determined by equation 5.14. For a layer L_k, the circle of confusion is denoted as d_k. The CoC for the in-focus layer is $d_{k_{\text{focus}}} = 0$ which means the layer in focus $(L_{k_{\text{focus}}})$ is not blurred. Starting from the in-focus layer, the CoC diameter increases symmetrically for layers closer and further away from the camera resulting in

$$d_{k_{\text{focus}}-i} = d_{k_{\text{focus}}+i} \tag{5.16}$$

when $(k_{\text{focus}} - i) \geq 0$ and $(k_{\text{focus}} + i) < K$.

The CoC d_{coc} is defined in units on the image sensor whereas a blurring method uses texture coordinates. With the following formula, the CoC can be defined in texture units:

$$d_{\text{pix}} = \frac{d_{\text{coc}} \cdot w_{\text{pix}}}{w}. \tag{5.17}$$

where w is the width of the image sensor and w_{pix} is the width of the rendering in pixel.

The width of the Gaussian filter kernel H_k, used for blurring layer L_k, is defined by σ_k (see section 3.4.1). Therefore, a conversion from the circle of confusion to the parameter of the blurring method (i.e., Gaussian filter) has to be established. This mapping is often chosen empirically or defined by a designer in other DoF methods (e.g., [34, 46, 68]). In this work the mapping is chosen empirically as

$$d_{\text{pix}} = 4\sigma. \tag{5.18}$$

A Gauss distribution within the range $[-2\sigma, 2\sigma]$ covers about 95% of the function's area. Due to the matting function, pixels contribute to more layers and therefore might be overblurred (section 5.2); pixels at a depth between anchor points z_{k-1} to z_k are present in layer L_k but also spread their values

to layers L_{k-1} and L_{k+1}. With equations 5.17 and 5.18, the CoC in image sensor coordinates can be determined by

$$d_k = \frac{4 \cdot \sigma_k \cdot w}{w_{\text{pix}}}. \tag{5.19}$$

With respect to the matting function, the following constraints for choosing σ_k should apply

$$\sigma_{k_{\text{focus}}-i} = \sigma_{k_{\text{focus}}+i}, \tag{5.20}$$

$$\sigma_{k_{\text{focus}}-i-1} > \sigma_{k_{\text{focus}}-i}, \tag{5.21}$$

$$\sigma_{k_{\text{focus}}+i+1} > \sigma_{k_{\text{focus}}+i}. \tag{5.22}$$

One concrete approach for defining σ_k is shown in section 5.5.

5.4 Blending

For the final compositing (I'), the blurred layers are blended together from back to front:

$$I' = L'_0 \oplus (L'_1 \oplus (\cdots \oplus (L'_{K-2} \oplus L'_{K-1}))). \tag{5.23}$$

Note that it is important to keep the right order for blending. Otherwise, hidden scene content (revealed by depth peeling) would be blended on top. The layer L'_{K-1} contains the background of the scene (i.e., pixels) with the highest depth value. In L'_0 are the pixels with the lowest depth value. To avoid blending and normalisation artefacts, the layers use premultiplied alpha values.

5.5 Optimisation: Cascading

The previous sections discussed the method of this thesis in four steps. This section proposes a method which combines the blurring and composition step (sections 5.3 and 5.4). Instead of blurring each layer separately, with a Gaussian filter kernel H_k, a cascaded approach is chosen. As discussed in section 3.4, recursively filtering with smaller filter kernels produces the same result as filtering with one bigger filter kernel. Although it is an advantage to use smaller filter kernels, simply using recursive filters would not be an optimisation because more filtering iterations would be needed. Thus, between each blurring iteration, one layer is blended onto the compositing before the blurring iteration continues. Each cascading filter kernel $\hat{H}_k$ is chosen so that the final result is the same as a single blur with kernel H_k.

Note that the front and the back layers, layers in front and behind the layer in focus respectively, have to be composed separately. Otherwise it is

not possible to compose the scene with the correct ordering of the layers. The composition of the front layer starts by taking the layer closest to the camera (i.e., L_0) and blurring it with the filter kernel $\hat{H}_0$. In the next step this blurred layer is blended, with $\oplus$, on the next closest layer (i.e., L_1) and afterwards blurred with $\hat{H}_1$. Since the blurred layer L_0 is over L_1 and then blurred again, the effect of this method is that L_0 is blurred by $\hat{H}_0$ and by $\hat{H}_1$. The iteration continues until the layer in-focus $L_{k_{\text{focus}}}$ is reached. Thus, the layers have been blurred recursively.

The back layers are blurred similarly starting with L_{K-1}. To keep the correct ordering of the layers, the layer closer to the camera (i.e., L_{K-2}) has to be blended over the previously blurred layer. The iteration is continued until the layer in-focus is reached.

The number of blurring iterations for a layer L_k is given by $|k - k_{\text{focus}}|$ and calculating the final composition I' is denoted as

$$I' = I'_{\text{front}} \oplus (L_{k_{\text{focus}}} \oplus I'_{\text{back}}), \tag{5.24}$$

where

$$I'_{\text{front}} = ((((L_0 * \hat{H}_0) \oplus L_1) * \hat{H}_1) \cdots \oplus L_{k_{\text{focus}}-1}) * \hat{H}_{k_{\text{focus}}-1}, \tag{5.25}$$

$$I'_{\text{back}} = (L_{k_{\text{focus}}+1} \oplus (\cdots (L_{K-2} \oplus (L_{K-1} * \hat{H}_{K-1})) * \hat{H}_{K-2})) * \hat{H}_{k_{\text{focus}}+1}. \tag{5.26}$$

This optimised composition approach delivers similar results to the composition in equation 5.23. However, there are differences in the final results which are further discussed in section 6.4.

The cascaded filter kernel $\hat{H}_k$ is a Gaussian filter kernel with a standard deviation of $\hat{\sigma}_k$. Results shown in chapter 6 use

$$\hat{H}_k = H_{\hat{\sigma}_k}, \tag{5.27}$$

where $\hat{\sigma}_k$ is defined as

$$\hat{\sigma}_k = |k - k_{\text{focus}}|. \tag{5.28}$$

Thus, σ_k needed for calculating the anchor points (equation 5.19), can be calculated by

$$\sigma_k = \begin{cases} 0 & \text{for} \quad k = k_{\text{focus}}, \\ \sqrt{\hat{\sigma}_k^2 + \sigma_{k+1}^2} & \text{for} \quad k < k_{\text{focus}}, \\ \sqrt{\hat{\sigma}_k^2 + \sigma_{k-1}^2} & \text{for} \quad k > k_{\text{focus}}, \end{cases} \tag{5.29}$$

where k is in the interval $[0, K-1]$. Recursive filters are discussed in section 3.4.4. Further information on cascaded Gaussians can be found in [17, chapter 8].

Chapter 6

Results and discussion

In this chapter, results produced with the method presented in chapter 5 are shown and discussed. The approach is implemented as an extension to the image manipulation program ImageJ.[1] Renderings (the scene I_0, peeled scene I_1, and their depth buffers) are supplied by an external program i.e., Blender. Results with different matting functions are shown in section 6.2. The spacing of the layers is discussed in section 6.3 and the impact of the used blurring method is described in section 6.4. Results with and without the occluded scene information are shown in section 6.5. The final method is compared to the ray tracer pbrt (section 6.6). For the comparisons and for the rendering results, a test scene is used. This scene is explained in the next section.

6.1 Test scene

The test scene `dof-dragons-v3`, used in this chapter, is a modification of the scene from [60] and used in [46]. A line of 11 dragons and a cone are placed in the scene, as shown from a top view, in figure 6.1. Due to the placement of the cone close to the camera, the scene is ideal for showing partial occlusion artefacts. Figures 6.2 (a) and (b) show pinhole renderings of `dof-dragons-v3` produced with pbrt and Blender respectively. Although the scene is the same in terms of positioning of objects and the camera, there are differences in the materials and lighting. This is due to the usage of two different stand alone applications (pbrt and Blender) and thus the different rendering methods. Furthermore, the renderings produced with Blender do not use anti-aliasing on edges because a pixel needs a single depth value for the matting. Antialiasing would result in interpolations in the depth buffer and in wrongly decomposed pixels.

[1] ImageJ is open source and can be downloaded from http://rsbweb.nih.gov/ij/.

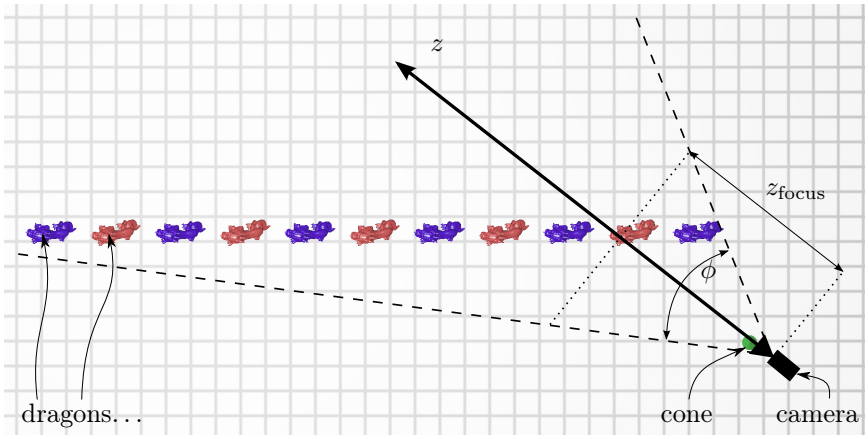
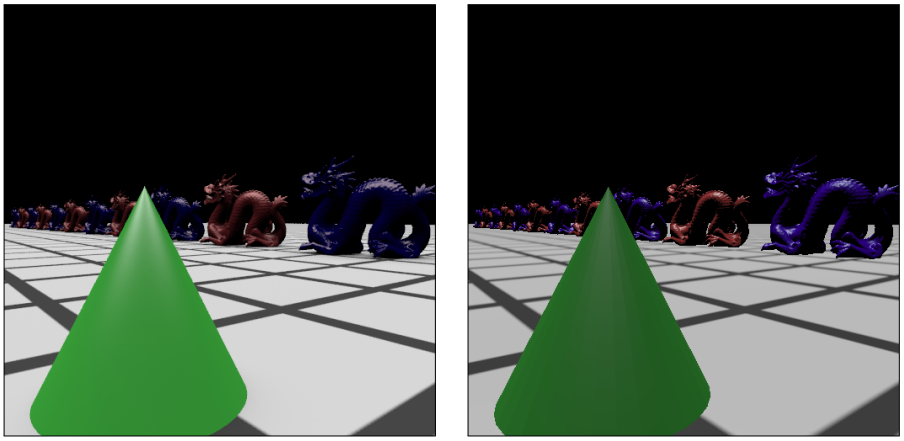


Figure 6.1: A top view of the test scene `dof-dragons-v3` used in this work, consisting of 11 dragons and a cone close to the camera to show the partial occlusion effect. The scene is a modification of a scene for pbrt [60].



(a) (b)

Figure 6.2: The test scene `dof-dragons-v3` rendered, with a pinhole model, in pbrt (a) and Blender (b). The field of view angle is set to $\phi = 60°$.

6.2 Layer matting

With the matting functions $\omega, \dot{\omega}$, and $\ddot{\omega}$ (discussed in section 5.2), a pixel contributes colour to three, two, and one layers with a weighted sum of 2, 1.5, and 1, respectively. However, the final composition does not get brightened up because of the over operator (section 3.3.2) and associated colours (section 3.3.3). If a rendered scene gets decomposed and composed again (without blurring) the brightness does not change.

The proposed matting function ω, as introduced in [46], prevents discreti-

sation artefacts known from [8]. In combination with depth peeling, however, a new problem arises: with the usage of the matting function ω for the decomposition of the peeled scene (equation 5.4) hidden pixels might be matted into a front layer. When the scene is finally composed, those hidden pixels are on top of foreground pixels resulting in artefacts. One such artefact is strongly visible in figure 6.3 (a) where the grey floor is visible on top of the red in-focus dragon. In figures 6.3 (a)–(d), the DoF range is large, therefore layers span across a wide depth range and thereby produce bigger discontinuities. If the layers have a smaller depth range, produced by more blur in the scene, this artefact is hardly visible (figure 6.4 (a)).

Using the matting function $\dot{\omega}$ (equation 5.9) prevents artefacts where peeled colour is blended on top of foreground pixels but introduces discretisation artefacts (known from [8]). In figure 6.3 (b), this issue is hardly noticeable. With bigger blurs, such as in figure 6.4 (b), the artefacts become more noticeable.

Using $\ddot{\omega}$ as a matting function produces discretisation artefacts in most situations. There are examples of this in figures 6.3 (c) and 6.4 (c).

One solution for the issue is to modify equation 5.4 to

$$L_k = \Big(I_0 \cdot \omega_k(Z_0)\Big) \oplus \Big(I_1 \cdot \dot{\omega}_k(Z_1)\Big), \tag{6.1}$$

which means the peeled information is matted with $\dot{\omega}$ while the scene rendering is matted with ω. The results, produced with this modified decomposition method, are free of the previously mentioned artefacts. This is shown in figure 6.3 (d) and figure 6.4 (d).

6.3 Layer anchor points

Since the proposed method of this work is a layered method, one important part is the decomposition of the scene into layers. A layer L_k contains pixels from I_0 and I_1 with a depth range from z_{k-1} to z_k (depending on the matting function, this range might be extended) where z_k and z_{k-1} are anchor points. In section 5.2 the calculation of the anchor points is discussed in detail. This section compares them to layer boundaries of other layered approaches, i.e., [46, 50].

Figure 6.5 shows a plot of $d_{\text{pix}}(z)$. The CoC d_{pix} is in pixel units and the parameters used for the plot are: $f = 0.05$, $N = 2.5$, $z_{\text{focus}} = 0.75$, $w = 0.058$, $w_{\text{pix}} = 512$. The layer bounds are symbolised by the dashed lines and the CoC for layer L_k, d_k are marked with circles. The number of layers is set to $K = 11$.

Note that all pixels with a depth greater than z_{K-1} use the same CoC d_{K-1}. The anchor points in the background ($z > z_{\text{focus}}$) are bound by equation 5.13. Therefore, d_{K-1} is a diameter which fulfils this constraint. Anchor points in the front ($z < z_{\text{focus}}$) are only bound by the number of layers K.

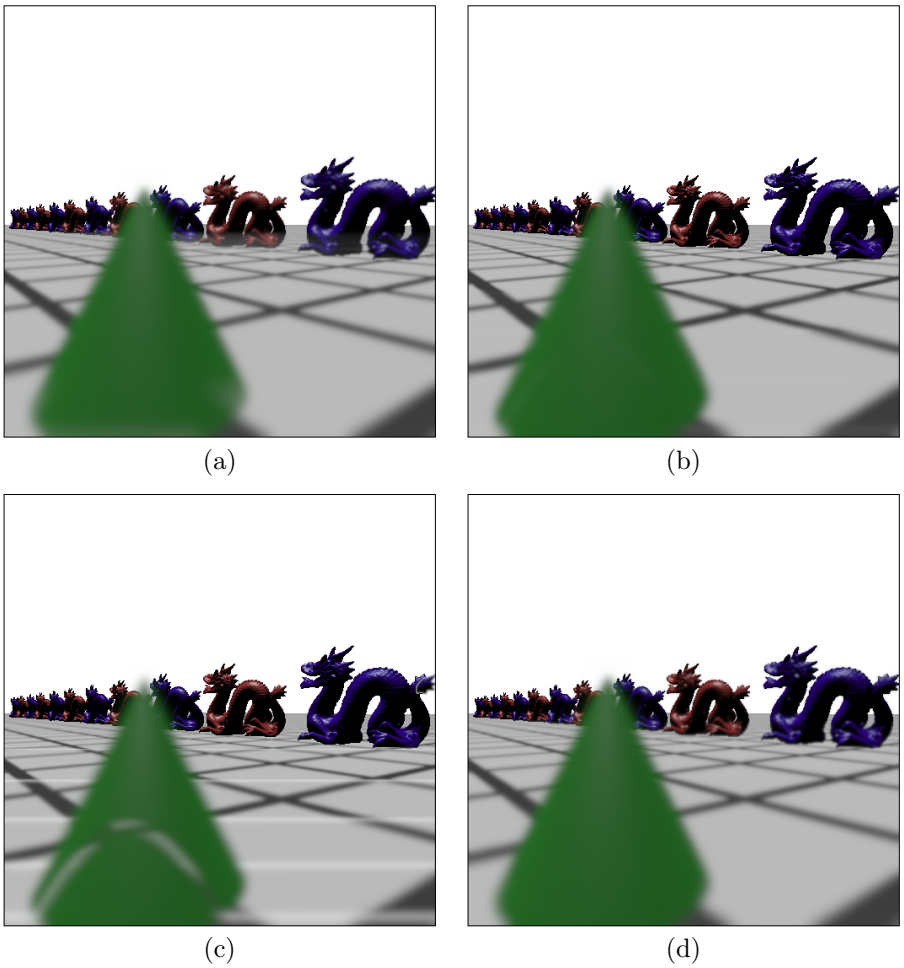


Figure 6.3: DoF renderings produced with the proposed method and the lens settings $N = 1.\overline{6}$, $\phi = 60°$, $f = 0.01$, and $w = 0.0114$. Results with the matting function ω (a), $\dot{\omega}$ (b), $\ddot{\omega}$ (c), and the advanced matting (d), from equation 6.1, are shown. Note that the matting function ω (a) produces artefact where the peeled scene bleeds through at the bottom of the green cone and on the red in-focus dragon. Weighting function $\ddot{\omega}$ (c) produces artefacts in objects distributed across layers where objects become partly transparent. This is especially noticeable on the green cone. Function $\dot{\omega}$ (b) and the advanced matting function (d) do not produce noticeable artefacts with the above settings.

Therefore K has to be chosen big enough so that the depth range of a scene is covered, because each anchor point further away from the focus plane at z_{focus} increases the maximum CoC diameter. If K is chosen too small, parts of the scene with big CoCs are not blurred enough for a realistic DoF effect. This problem is shown in figure 6.6 (a) where $K = 5$ produces too little

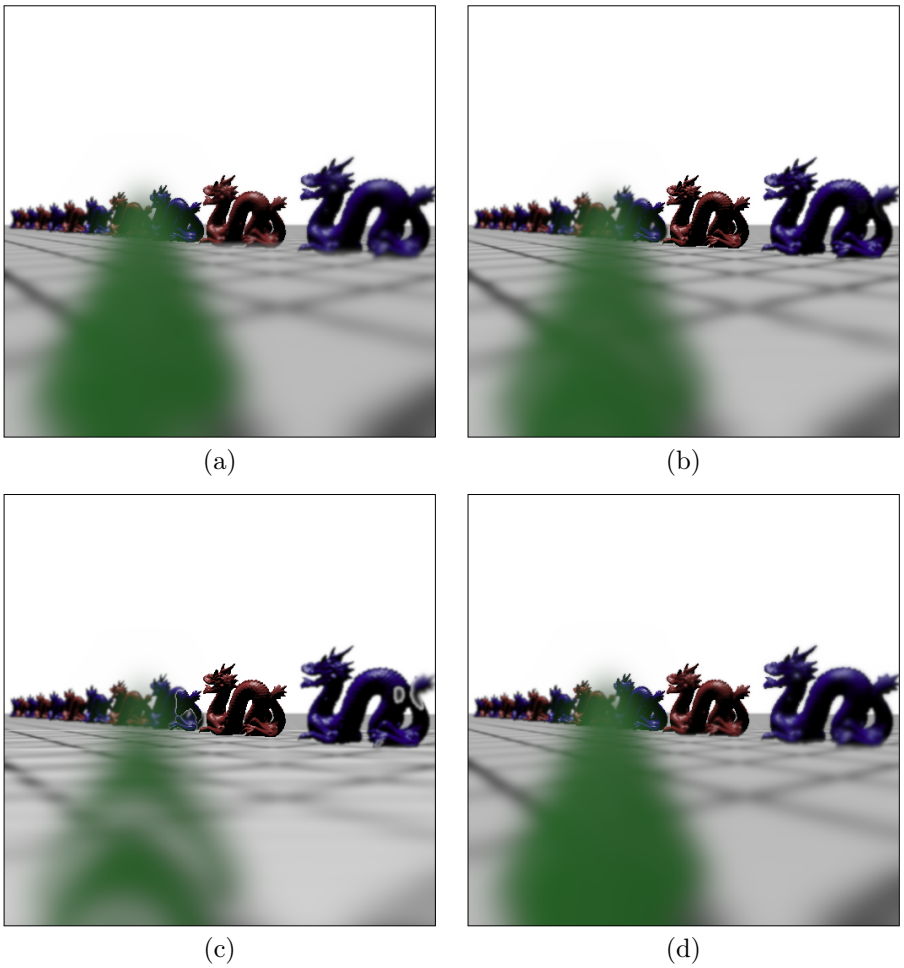


Figure 6.4: DoF renderings produced with the proposed method and the lens settings $N = 0.5$, $\phi = 60°$, $f = 0.01$, and $w = 0.0114$. Similar to figure 6.3, the results with the matting function ω (a), $\dot{\omega}$ (b), $\ddot{\omega}$ (c), and the advanced matting (d), from equation 6.1, are shown. Similar to figure 6.3, the matting function $\ddot{\omega}$ (c) produces noticeable artefacts. At higher blur radii, $\dot{\omega}$ (b) results in ringing artefacts this is slightly visible on the green cone. With the above settings, ω (a) and the advanced matting function (d) produce results with no noticeable artefacts.

blur for the cone at the front. The parameters used for these renderings are $\phi = 60°$, $N = 0.5$, $f = 0.01$, and $w = 0.0114$. Figure 6.6 (b) shows a sufficient amount of layers. The cone in the scene `dof-dragons-v3` is located at depth $z_{\text{cone}} = 0.10$ and $z_0 = 0.378$ with $K = 5$. Therefore, the pixels from the cone get blurred with d_0, which is an indicator for too little blurring. Increasing K to $K = 15$ layers is enough for the same scene because then $z_0 = 0.084$, $z_1 = 0.094$, and $z_2 = 0.107$. Thus, a pixel at depth $z_{\text{cone}} = 0.10$

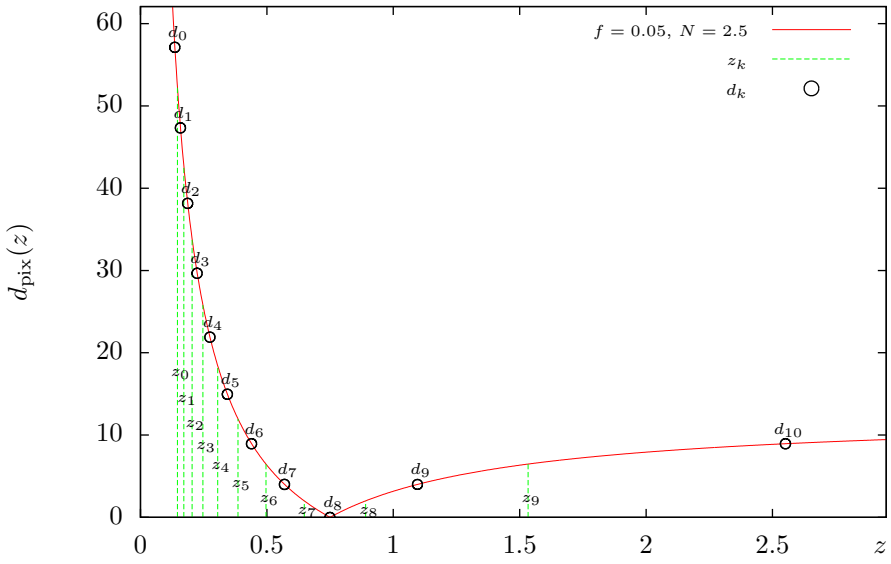
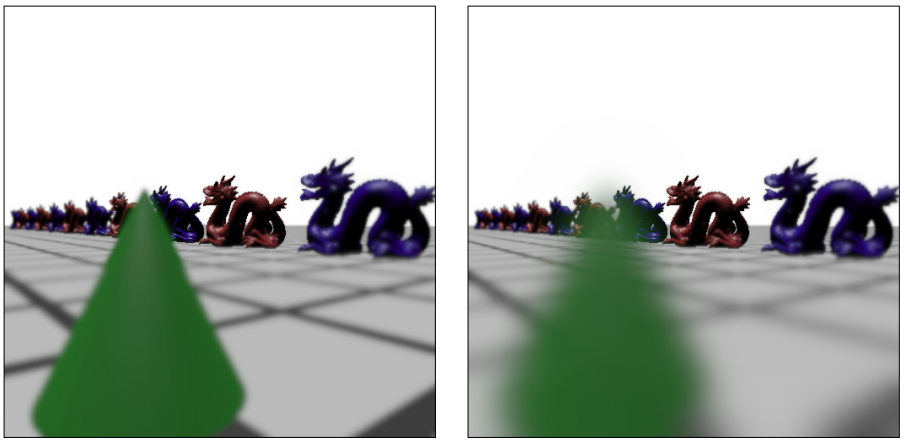


Figure 6.5: A plot showing $d_{\text{pix}}(z)$, the anchor points z_k, and the CoCs d_k for the layers with $K = 11$. The parameter settings are: $f = 0.05$, $N = 2.5$, $z_{\text{focus}} = 0.75$, $w = 0.058$, $w_{\text{pix}} = 512$.



(a) (b)

Figure 6.6: The test scene `dof-dragons-v3` rendered with the proposed method where $K = 5$ (a) and $K = 30$ (b). The settings used are $\phi = 60°$, $N = 0.5$, $f = 0.01$, and $w = 0.0114$. Note that a too small amount of layers as shown in (a) produces too little blur for front objects: the green cone and parts of the floor.

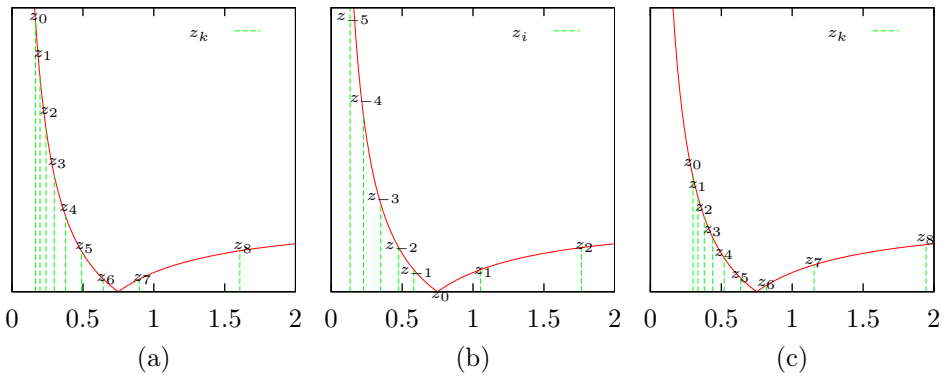


Figure 6.7: The anchor points of two layered DoF methods plotted as in figure 6.5 but with less details. A plot of the method proposed in this work with the parameters $K = 9$, $\phi = 60°$, $N = 0.5$, $f = 0.01$, $w = 0.0114$, and $w_{\text{pix}} = 512$ (a). The splitting of the layers in [46], with $r = 0.01$, is shown in (b). The Layer boundaries of [50] are shown in (c) with similar settings as in (a) along with $z_{\text{near}} = 0.3$.

will be blurred with the CoC diameter d_2.

The layer splitting in this work strongly relates to the used blurring method. Other layered DoF methods, as discussed in section 4.2.3, i.e., [46, 50], also decompose the layers by defining anchor points. A comparison of anchor points for layered methods are shown in figures 6.7 (a)–(c) where this work's methods, [46] and [50], are ploted with similar settings.

The method proposed in [46] covers high blur radii near the camera with less layers because of the used pyramid blurring method. In [50] z_{near}, a user-defined variable, is limiting the maximum CoC to $d_{\text{coc}}(z_{\text{near}})$.

6.4 Blurring methods

One optimisation discussed in section 5.5, is the use of cascaded Gaussian filters and the combination of blending and blurring. With this optimisation the blurring width of the filter kernel, σ of a Gaussian distribution, for each layer can be reduced. With the usage of filters on scaling levels, such as done in [46], this approach could be optimised more. Further information on filtering on scaling levels can be found in section 3.4. However, this is not implemented in this work and would reduce the quality of filtering.

Although recursively filtering with a Gaussian distribution produces similar results as filtering with one big Gaussian kernel, the blending step between each recursion introduces differences in the results. The comparisons between figures 6.8 (a) to (b) and figures 6.8 (c) to (d) show those differences. One explanation for the differences in the results is as follows: before

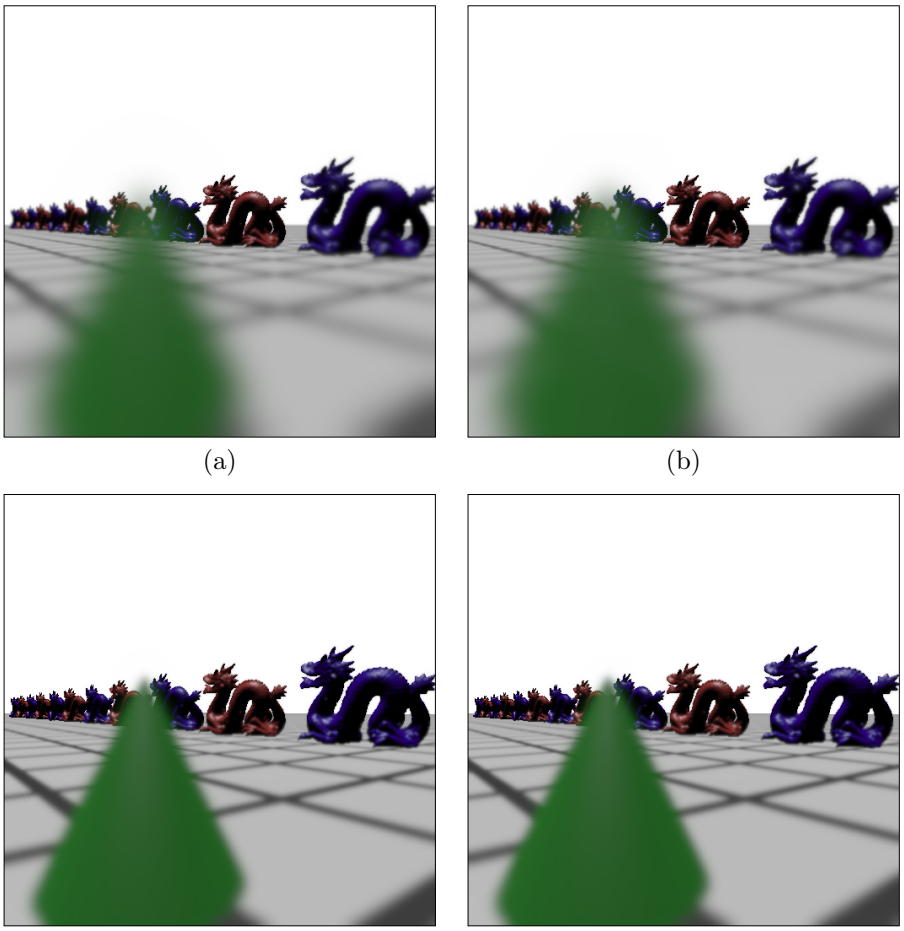


Figure 6.8: A comparison of the cascaded Gaussian or the regular Gaussian blurring. The DoF effect generated with the settings $N = 0.5$, $\phi = 60°$, $f = 0.01$, and $w = 0.0114$ is shown with the cascaded (a) and the Gaussian blurring (b). The rendering with a different f-stop of $N = 1.\overline{6}$ is shown with cascaded (c) and with Gaussian blurring (d). Note that there are slight differences in the results when using the cascaded approach compared to the Gaussian blur, especially if the blurring radii in the scene are large, such as in (a) and (b).

blurring a layer L_k (in the cascaded approach), the previously blurred and composed layers $L_{k-1}, \ldots, L_0$ are blended on top of the current layer. Thus, L_k contributes less to the next blurring step, because some information is not existent anymore or it's intensity is reduced.

Nevertheless, for example, after 10 filtering iterations with the recursive Gaussian approach, the width of the recursive Gaussian kernel is still $\hat{\sigma}_{10} = 10.00$ while the single Gaussian blur needs a width of $\sigma_{10} = 19.62$ (as

i	1	2	3 ... 9	10	11	12	13	14 ...
$\hat{\sigma}_i$	1.00	2.00	3.00 ... 9.00	10.00	11.00	12.00	13.00	14.00 ...
σ_i	1.00	2.24	3.74 ... 16.88	19.62	22.49	25.50	28.62	31.86 ...

Table 6.1: A table showing $\hat{\sigma}$ and the resulting σ if the filter kernels $H_{\hat{\sigma}}$ are applied recursively. The number of recursion is denoted by i.

shown in table 6.1). The first few filtering iterations only slightly blur the results stronger. Therefore, scenes with high blur radii gain higher performance increases because the convolution with a single big Gaussian kernel is avoided.

6.5 Depth peeling

One essential part of this thesis, to avoid partial occlusion artefacts, is depth peeling (section 5.1). Depth peeling suffers similar problems as shadow mapping. Therefore, a good choice of the bias value ϵ is essential. Furthermore, rendering methods such as backface culling should be enabled to avoid the pollution of the peeled scene with backside polygons. The implementation of depth peeling in a rendering pipeline needs changes in all materials-shaders used in a scene, and therefore requires a lot of changes in simple rendering pipelines.

The peeled scene and peeled depth for the scene `dof-dragons-v3` is produced with two additional renderings and their depth buffers; where one rendering is without the green cone and the second without the cone and the dragons. Therefore, the renderings produced with Blender do not contain shadows or reflections (which would cause artefacts). The scene, the peeled scene, and their depth buffers are shown in figures 5.2 (a)–(d).

If the peeled scene information (I_1 and Z_1) is not used for matting, the approach reduces to a simple layered DoF method producing discretisation artefacts. Figures 6.9 (a) and (b) show renderings *without depth peeling*. With small CoCs, thus small blur radii as in figure 6.9 (a), the artefact is less noticeable. Big CoCs, as in figure 6.9 (b), produce noticeable artefacts at the borders of front objects. Figures 6.9 (c) and (d) show the DoF effect with the same parameters but *with depth peeling* enabled.

6.6 Ray tracing with pbrt

As discussed in chapter 4, a ray tracer produces optically accurate DoF effects and also does not suffer from partial occlusion problems. Therefore, the test scene is rendered with the ray tracer pbrt from [60]. In pbrt the configurable camera parameters are the field of view ϕ, the focus plane z_{focus}, and the radius of the lens r, where $a = 2r$.

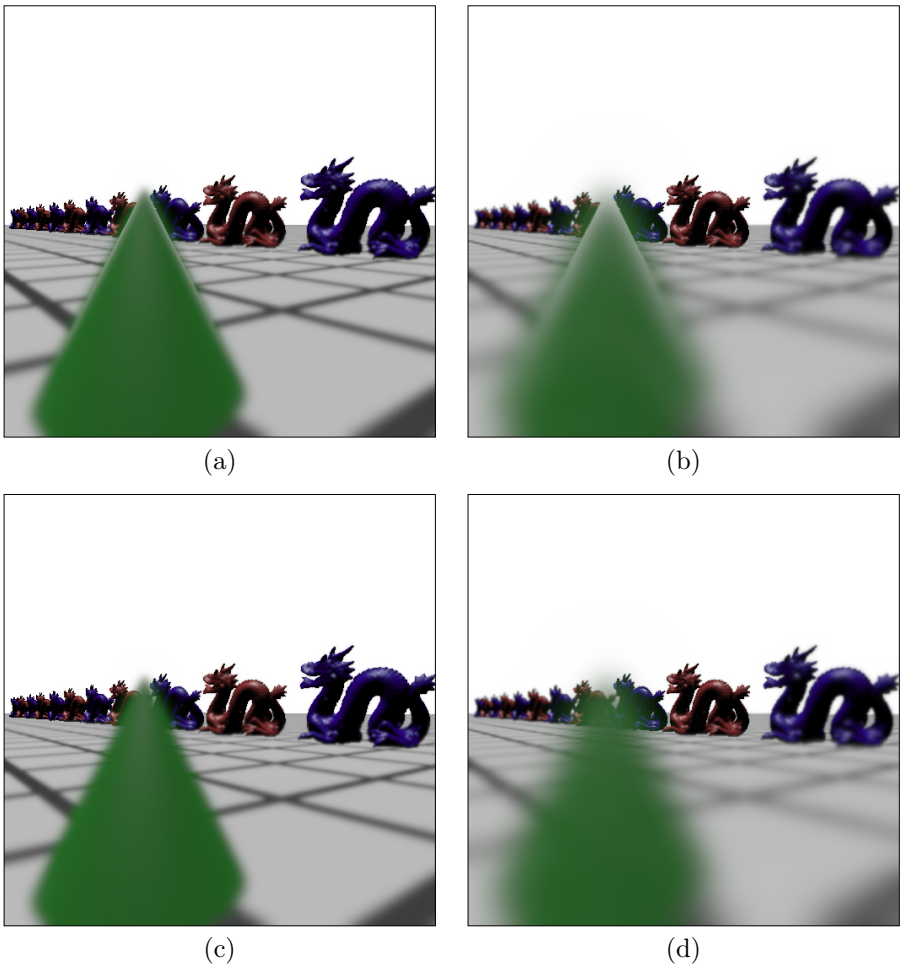


Figure 6.9: The Dof effect, produced with the method discussed in chapter 5 *without* the peeled scene information (I_1 and Z_1), with parameters $N \approx 0.5$ (a) and $N = 1.67$ (b). The compositions *with* depth peeling (c) and (d) respectively. The focal length is set to $f \approx 0.01$ and the image sensor width is $w = 0.0114$. Note that for scenes with small blur radii, such as (a) and (c), the missing of peeled scene information is less noticeable. In scenes with higher CoCs, shown in (b) and (d), partial occlusion artefacts are visible; the artefact is strongly visible at the top of the green cone where dragons should be partly visible.

To understand the mapping of pbrt-parameters to lens parameters, some basics of pbrt are discussed. If a pinhole camera model is simulated in pbrt, a ray starts from the origin $\mathsf{p}_0 = (0, 0, 0)$ and passes a point on the view plane p_{view}. This is shown in figure 6.10. When simulating a finite aperture, the origin of a ray gets altered to any position on the lens, distributed by a point spread function, where the lens is defined by the aperture. Instead of

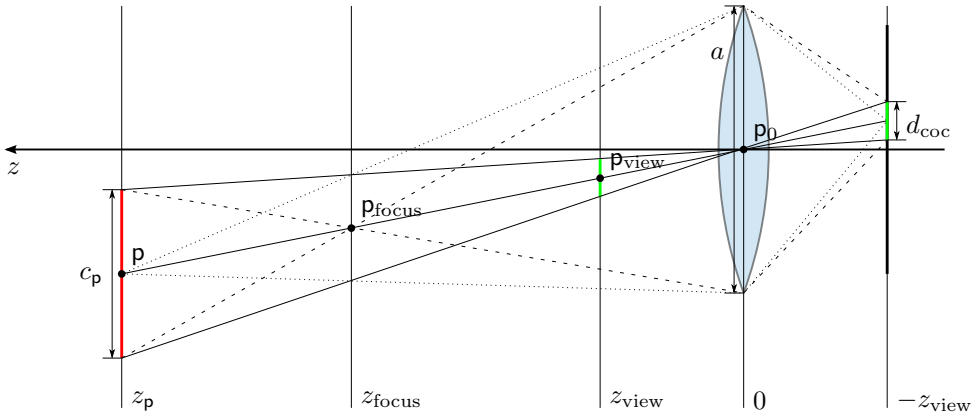


Figure 6.10: A schemata to show how the ray tracer pbrt produces depth of field.

calculating snell's law (equation 2.1), pbrt uses the constraint that all rays from the lens pass one point $\mathsf{p}_{\text{focus}}$ located at the focus plane at z_{focus}. The point $\mathsf{p}_{\text{focus}}$ can be calculated with the vector $\mathsf{p}_0\vec{\mathsf{p}}_{\text{view}}$ (figure 6.10). With $\mathsf{p}_{\text{focus}}$, the new direction of the ray, $\mathsf{p}_0\vec{\mathsf{p}}_{\text{focus}}$, can be computed.

With similar triangles (dashed lines in figure 6.10), the following can be shown:

$$\frac{c_{\mathsf{p}}}{z_{\mathsf{p}} - z_{\text{focus}}} = \frac{a}{z_{\text{focus}}}. \tag{6.2}$$

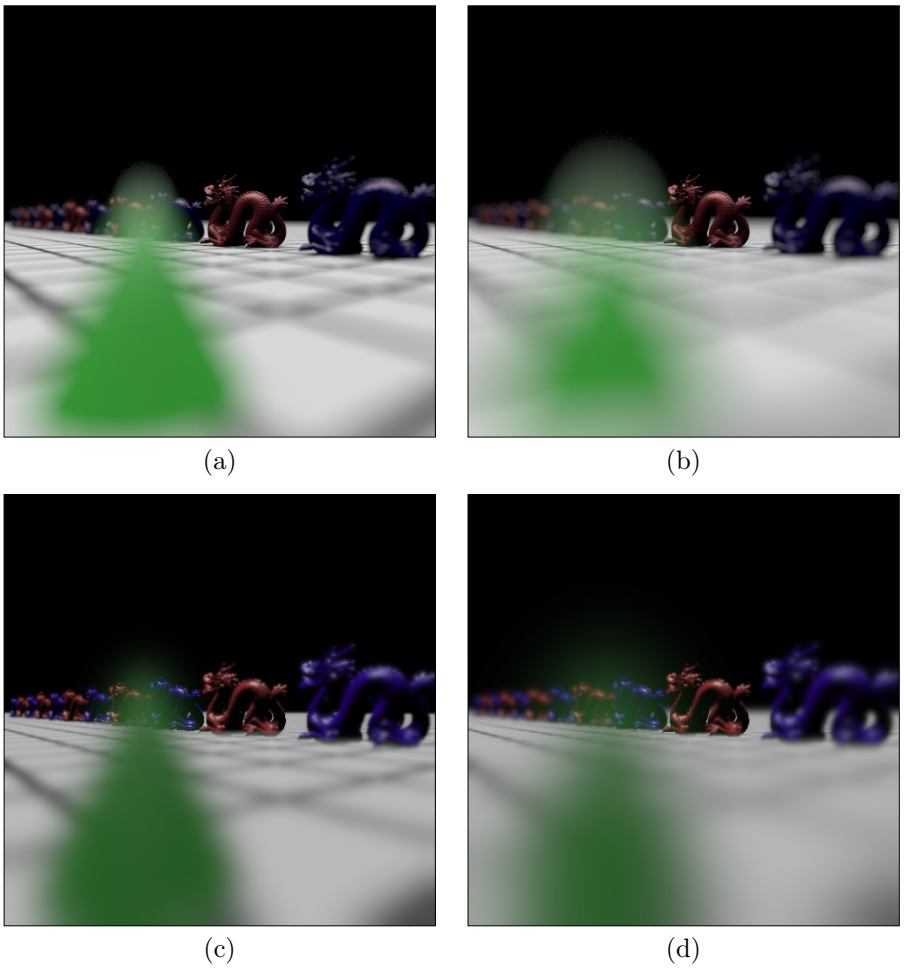
Thus, the circle of confusion can be obtained by

$$\frac{d_{\text{coc}}}{z_{\text{view}}} = \frac{c_{\mathsf{p}}}{z_{\mathsf{p}}}, \tag{6.3}$$

shown with solid lines in figure 6.10. In [46], the parameters of pbrt are used to generate the DoF effect. Parameters, which are typically better known by photographers (f, N, w as discussed in chapter 2), are used in this work. One important parameter, for the conversion between pbrt and lens parameters, is the location of the view plane at z_{view}. In pbrt $z_{\text{view}} = z_{\text{near}}$, where the view plane is set to $z_{\text{view}} = 0.01$ in the programme source code. Since the view plane distance also defines the distance of the image plane, with equation 2.14 the focal length f can be calculated. Thus, w and N can be calculated, with equations 3.4 and 2.10 respectively.

Comparison of renderings

After establishing the relation between pbrt-parameters and DoF-parameters, renderings are compared. Figures 6.11 (a) and (b) show renderings of `dof-dragons-v3` produced with pbrt and different lens diameters $r = 0.01$ and



(a) (b) (c) (d)

Figure 6.11: Test scene `dof-dragons-v3` rendered in pbrt with a lens radius of $r = 0.01$ (a) and $r = 0.02$ (b). The field of view angle is set to $\phi = 60°$. Dof effects produced with the method discussed in chapter 5 with parameters $f = 0.00987$, $w = 0.0114$, $N = 0.493$ (c), and $N = 0.247$ (d). Note that the ray tracer's PSF differs from this work's PSF (Gaussian). Furthermore the ray tracer uses the entire scene description, while the post processing method only uses pinhole renderings; noticeable at the bottom of the green cone, where the ray adds grey colour from the surrounding floor while the proposed method does not.

$r = 0.02$. The field of view is set to $\phi = 60°$ and the focus plane is at $z_{\text{focus}} = 0.75$. The resolution of the renderings is 512×512 resulting in $w_{\text{pix}} = 512$. From the parameters defined in pbrt, the focal length f can be computed by

$$f = \frac{z_{\text{view}} \cdot z_{\text{focus}}}{z_{\text{view}} + z_{\text{focus}}}, \tag{6.4}$$

resulting in $f = 0.00987$ or $f \approx 0.01$. For the conversion of the CoC between screen and sensor units, the image sensor size w is needed. By rearranging equation 3.4 to

$$w = 2f \tan\left(\frac{\phi}{2}\right), \tag{6.5}$$

the width can be calculated resulting in $w = 0.0114$.

With the above calculated parameters this work's DoF method can be used. Examples of renderings are shown in figures 6.11 (c) and (d) with $N = 0.493$ and $N = 0.247$ respectively. The scene I_0 and peeled scene I_1 along with the depth buffers Z_0 and Z_1 are renderings produced with Blender.

The renderings produced with this work and the results of the ray tracer look similar. Partial occlusion, especially at the green cone, is handled similarly and objects in the scene are blurred similarly strong. However, since the renderings are produced with different programs, it is not comparable by any image similarity method. Furthermore, the point spreading function used in pbrt does not match the Gaussian shape used in this work.

Additionally, post-processing methods as used in this thesis produce different results at image borders. A ray tracer uses the full scene description, while the post-processing DoF only uses the already rendered scene. This problem is apparent in figures 6.11 (a) and (b) at the bottom of the green cone, where the ray traced images are blurred to grey, due to the surrounding grey floor. While the post-processed results do not show this effect (figures 6.11 (c) and (d)); one solution, minimizing this artefact, is to render a bigger viewport of the scene and crop it after post-processing.

Chapter 7

Conclusion and outlook

In this work, an approach for solving the partial occlusion problem in depth-of-field post-processing effects is presented. The optical effects leading to DoF, and techniques and processes in computer renderings have been discussed in chapters 2, and 3. Current methods for generating DoF effects on artificially generated images have been presented, in a chronological order, in chapter 4. Problems and strengths of previous methods, with respect to partial occlusion, have been discussed.

The novel contribution of this work, discussed in chapter 5, is the combination of DoF methods (mainly [46, 49, 50]). This work's method uses layers, as done in other DoF approaches (i.e., [8, 39, 46, 50]), where the layer spacing is inspired by [46]. Hidden scene information, generated with depth peeling, is used. One downside of depth peeling is the costly integration into existing rendering pipelines. Layered rendering, as done in [50], might be an alternative but comes at additional rendering costs.

Each layer can be blurred uniformly by gathering filters, well suited for GPUs. The current implementation uses Gaussian filters (a PSF frequently used in DoF methods e.g., [46, 51, 74]) because Gaussians can be applied recursively. Combining the filtering and composing steps as an optimisation, to reduce the kernel size for the layer blurring, alters the result (when compared to individual layer blurring) but reduces the filtering effort. The filtering effort for high blurriness in the scene is rising linearly because of the separability of Gaussian filters. Since each layer is blurred uniformly, any other filtering method can be used if the optimisation is left out.

The composition method used in chapter 5 uses alpha blending for combining the layers, which is simpler than [50], but also avoids normalization issues of [49]. Layering discretisation artefacts known from other methods are avoided by the matting function from [46].

The current implementation is not optimized for interactive rates. Thus, this method is suited for generating DoF effects in previously rendered scenes where depth peeling is possible. Producing DoF effects with a ray tracer is

physically accurate but comes at high rendering times. One application for this work's method is to generate DoF effects on pinhole renderings produced with a ray tracer or another accurate rendering method. However, it is unusual and needs additional considerations to produce a depth buffer and a peeled scene; especially with a ray tracer.

As discussed in chapter 5, the antialiasing of edges (e.g., MSAA) cannot be enabled for scene renderings because a pixel should not contain interpolated colours or interpolated depth values. With the rising popularity of deferred shading in rendering pipelines, image based antialiasing methods (such as MLAA) are more popular. Therefore, there are methods available to antialias renderings as a post-process.

Future work

For future work, the usage of the presented approach on animated scenes can be examined. With the usage of layers, there might be popping artefacts[1] caused by the layer discretisation. Although such artefacts should be reduced by the smoothness of the matting function, the impact of this work's approach still has to be verified.

With the usage of cascaded Gaussian filters, the filtering process is optimized. There are further possibilities to optimize the blurring of the layers. Depending on the application, approximations of a Gaussian might be sufficient. Therefore, scale based methods (e.g., [45]) can be used. If speed is a high priority goal, mipmapping and stochastic sampling as in [68] are options for the filtering method. Since each layer is blurred uniformly, other uniform filtering approaches different than recursive Gaussians can be applied. Options for such blurring methods are filters in the frequency domain or filter spreading, as in [42]. Furthermore, a filtering method with the possibility to change the PSF would allow to simulate various shapes of aperture stops.

The layer decomposition could be optimized by using clustering methods, such as k-means clustering, as proposed in [40, 49]. With the use of clustering, layer borders could be tailored to the pixel density in scenes and empty layers could be avoided. However, clustering is a costly process and therefore only applicable for off-line rendering.

Furthermore, the presented approach should be implemented on modern graphics hardware using OpenGL or DirectX for performance benchmarks. The author of this work designed this approach for interactive rendering rates. Thus, especially when the blurring method is changed to an inaccurate method and the number of layers is kept low, this should be an achievable goal.

[1] Popping artefacts appear in animated sequences. The word popping refers to the fact that objects or pixels appear or disappear in a short time period (often one frame) without any form of smooth transition. Thus, the abrupt change is recognised as artefact.

Appendix A

Content on CD-ROM

Standard: CD-ROM, single layer, ISO9660

A.1 Thesis

Path: /

thesis.pdf	David C. Schedl's master thesis

A.2 Source code

Path: /source/

Doflj/	source code for ImageJ plugins
exr2txt/	source code of a tool for converting `*.exr` images to `*.txt` files, needed for the ImageJ plugins

A.3 Scene files

Path: /scenes/

blender-scenes/	test scenes for Blender
pbrt-scenes/	test scenes for pbrt

A.4 Thesis figures

Path: /images/

cha_optics/	figures used in chapter 2
cha_postpro/	figures used in chapter 3
cha_previous/	figures used in chapter 4

cha_method/	figures used in chapter 5
cha_results/	figures used in chapter 6
cha_*/*.png	images used in the thesis
cha_*/*.pdf	PDF-files containing figures
cha_*/*.plot	raw files for generating plots with gnuplot
cha_*/*.svg	raw files of figures in an editable vector format

A.5 Papers

Path: /papers/

*.pdf	papers cited and discussed in the thesis

A.6 Applications

Path: /applications/

pbrt-v2/	contains executable files for pbrt and auxiliary tools
blender*	installation files for Blender

Abbreviations

CoC	circle of confusion
CPU	central processing unit
DoF	depth of field
FPS	frames per second
GPU	graphics processing unit
HDR	high dynamic range
LUT	lookup table
MLAA	morphological antialiasing
MRT	multiple render target
MSAA	multisample antialiasing
PSF	point spread function

Bibliography

[1] T. Akenine-Moeller, J. Munkberg, and J. Hasselgren. Stochastic rasterization using time-continuous triangles. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 7–16, Aire-la-Ville, Switzerland, 2007. ACM.

[2] T. Akenine-Moeller, E. Haines, and N. Hoffman. *Real-Time Rendering*. A K Peters, 3^{rd} edition, 2008.

[3] A. C. Barkans. High quality rendering using the talisman architecture. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 79–88, New York, USA, 1997. ACM.

[4] B. A. Barsky and T. J. Kosloff. Algorithms for rendering depth of field effects in computer graphics. In *Proceedings of the 12th WSEAS international conference on Computers*, pages 999–1010, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).

[5] B. A. Barsky, D. R. Horn, S. A. Klein, J. A. Pang, and M. Yu. Camera models and optical systems used in computer graphics: Part I, Object based techniques. Technical report, University of Berkeley, California, USA, 2003. URL http://graphics.cs.berkeley.edu/papers/Barsky-CMO-2003-05/.

[6] B. A. Barsky, D. R. Horn, S. A. Klein, J. A. Pang, and M. Yu. Camera models and optical systems used in computer graphics: Part II, Image-based techniques. Technical report, University of Berkeley, California, USA, 2003. URL http://graphics.berkeley.edu/papers/Barsky-CMP-2003-05/.

[7] B. A. Barsky, D. R. Tobias, Michael J.and Horn, and D. P. Chu. Investigating occlusion and discretization problems in image space blurring techniques. In *First International Conference on Vision, Video and Graphics*, pages 97–102, University of Bath, UK, July 2003.

[8] B. A. Barsky, M. J. Tobias, D. P. Chu, and D. R. Horn. Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques. *Graphics Models*, 67(6):584–599, Nov. 2005.

[9] M. Bertalmio, P. Fort, and D. Sanchez-Crespo. Real-time accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, 3DPVT '04, pages 767–773, Washington, DC, USA, 2004.

[10] K. Bjorke. High-quality filtering. In F. Randima, editor, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter 24. Pearson Education, Amsterdam, 2004.

[11] J. Blinn. *Jim Blinn's Corner: A Trip Down the Graphics Pipeline*. Morgan Kaufmann, 1996.

[12] J. Blinn. *Jim Blinn's Corner: Dirty Pixels*. Morgan Kaufmann, 1998.

[13] J. Blow. Mipmapping, part 1. online, Dec. 2001. URL http://number-none.com/product/Mipmapping,%20Part%201/index.html.

[14] J. Blow. Mipmapping, part 2. online, Jan. 2002. URL http://number-none.com/product/Mipmapping,%20Part%202/index.html.

[15] R. Bridson. Fast poisson disk sampling in arbitrary dimensions. Technical report, University of British Columbia, 2007. URL http://people.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf.

[16] W. Burger and M. J. Burge. *Digital Image Processing An Algorithmic Introduction using Java*. Springer, 2008.

[17] W. Burger and M. J. Burge. Principles of Digital Image Processing: Advanced Techniques. To appear, 2011.

[18] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. In R. Wolfe, editor, *Seminal graphics*, pages 183–188. ACM, 1998.

[19] R. L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics (TOG)*, 5:51–72, Jan. 1986.

[20] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 137–145, New York, NY, USA, 1984. ACM.

[21] F. C. Crow. Summed-area tables for texture mapping. *SIGGRAPH Computer Graphics*, 18:207–212, Jan. 1984.

[22] J. Demers. Depth of field: A survey of techniques. In F. Randima, editor, *GPU Gems*, chapter 23, pages 375–390. Pearson Education, 2004. URL http://http.developer.nvidia.com/GPUGems/gpugems_ch23.html.

[23] M. A. Z. Dippé and E. H. Wold. Antialiasing through stochastic sampling. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 69–78, New York, USA, 1985.

[24] J. Donald. The ultimate depth-of-field skinny. online, Nov. 2002. URL http://www.dvinfo.net/articles/optics/dofskinny.php.

[25] W. Engel, editor. *ShaderX*^{7} *– Advanced Rendering Techniques*. Charles River Media, 2009.

[26] W. Engel, editor. *GPU Pro – Advanced Rendering Techniques*. ShaderX Book Series. A K Peters, 2010.

[27] C. Everitt. Interactive order-independent transparency. Technical report, NVIDIA, 2001. URL http://developer.nvidia.com/content/interactive-order-independent-transparency.

[28] P. Fearing. Importance ordering for real-time depth of field. In *Proceedings of the Third International Conference on Computer Science*, pages 372–380, Hong Kong, 1996.

[29] J. D. Foley, A. v. Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics Principles and Practice*. Addison-Wesley, 1997.

[30] G. R. Fowles. *Introduction to Modern Optics*. Dover Publications, New York, 2^{nd} edition, 1989.

[31] A. Glassner. Adaptive precision in texture mapping. *SIGGRAPH Computer Graphics*, 20:297–306, Aug. 1986.

[32] N. Goldberg. *Camera Technology: The Dark Side of the Lens*. Academic Press, 1992.

[33] P. Haeberli and K. Akeley. The accumulation buffer: hardware support for high-quality rendering. *SIGGRAPH Computer Graphics*, 24:309–318, Sept. 1990.

[34] E. J. Hammon. Practical post-process depth of field. In H. Nguyen, editor, *GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 28, pages 583–606. Addison-Wesley, 2007. URL http://http.developer.nvidia.com/GPUGems3/gpugems3_ch28.html.

[35] P. S. Heckbert. Filtering by repeated integration. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 315–321, New York, 1986. ACM.

[36] P. S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6:56–67, Nov. 1986.

[37] P. S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1989.

[38] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. *Computer Graphics Forum*, 24:547–555, 2005.

[39] M. Kass, L. Aaron, and J. Owens. Interactive depth of field using simulated diffusion on a GPU. Technical report, Pixar Animation Studios, 2006. URL http://graphics.pixar.com/DepthOfField/paper.pdf.

[40] T. J. Kosloff. *Fast Image Filters for Depth of Field Post-Processing*. PhD thesis, EECS Department, University of California, Berkeley, May 2010. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-69.html.

[41] T. J. Kosloff and B. A. Barsky. An algorithm for rendering generalized depth of field effects based on simulated heat diffusion. In *Proceedings of the 2007 International Conference on Computational Science and its Applications, ICCSA'07*, pages 1124–1140, Kuala Lumpur, Malaysia, 2007. Springer.

[42] T. J. Kosloff, J. Hensley, and B. A. Barsky. Fast filter spreading and its applications. Technical report, EECS Department, University of California, Berkeley, Apr. 2009. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-54.html.

[43] T. J. Kosloff, M. W. Tao, and B. A. Barsky. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proceedings of Graphics Interface 2009*, pages 39–46, Toronto, Canada, 2009.

[44] M. Kraus. Quasi-convolution pyramidal blurring. In *Proceedings of the Third International Conference on Computer Graphics Theory and Applications*, GRAPP 08, pages 155–162, Funchal, Madeira, Portugal, 2008.

[45] M. Kraus and M. Strengert. Pyramid filters based on bilinear interpolation. In *Proceedings of the Second International Conference on*

Computer Graphics Theory and Applications, GRAPP 2007, pages 21–28, Barcelona, Spain, Mar. 2007. INSTICC – Institute for Systems and Technologies of Information, Control and Communication.

[46] M. Kraus and M. Strengert. Depth-of-field rendering by pyramidal image processing. *Computer Graphics Forum*, 26(3):645–654, 2007.

[47] J. Kriv´anek, J. Zára, and K. Bouatouch. Fast depth of field rendering with surface splatting. In *Computer Graphics International, CGI 2003*, pages 196–201, Tokyo, Japan, July 2003. IEEE Computer Society.

[48] W. Lance. Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, pages 270–274, Atlanta, Georgia, USA, Aug. 1978. ACM.

[49] S. Lee, G. J. Kim, and S. Choi. Real-time depth-of-field rendering using splatting on per-pixel layers. *Computer Graphics Forum (Proc. Pacific Graphics'08)*, 27(7):1955–1962, 2008.

[50] S. Lee, E. Eisemann, and H.-P. Seidel. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics (TOG)*, 28(5): 1–6, 2009.

[51] S. Lee, G. J. Kim, and S. Choi. Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):453–464, 2009.

[52] S. Lee, E. Eisemann, and H.-P. Seidel. Real-time lens blur effects and focus control. *ACM Transactions on Graphics (TOG)*, 29(4):65:1–65:7, July 2010.

[53] J. McCormack, J. M, K. I. Farkas, N. P. Jouppi, and R. Perry. Simple and Table Feline: Fast Elliptical Lines for Anisotropic Texture Mapping. Technical report, Western Research Laboratory, Palo Alto, California, USA, Oct. 1999. URL http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-99-1.pdf.

[54] J. L. Michtell, M. Y. Ansary, and E. Hart. Advanced Image Processing with DirectX9 Pixel Shaders. In F. W. Engel, editor, *ShaderX²: Shader programming Tips and Tricks With DirectX 9*, chapter 4, pages 439–464. Wordware Publishing, Plano, Texas, USA, 2004.

[55] L. Moore. The heat equation and diffusion. Slides, Macquarie University, Sydney, Australia, 2004. URL http://www.physics.mq.edu.au/~wardle/PHYS220/heat_eqn.ppt.

[56] J. D. Mulder and R. van Liere. Fast perception-based depth of field rendering. In *Proceedings of the ACM symposium on Virtual Reality Software and Technology*, VRST '00, pages 129–133, Seoul, Korea, Oct. 2000. ACM.

[57] H. Nguyen, editor. *GPU Gems 3: Programming Techniques for High-Performance Graphics and General-Purpose Computation.* Addison-Wesley, 2007.

[58] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, July 1990.

[59] M. Pharr, editor. *GPU Gems 2: Techniques for Graphics and Compute-Intensive Programming.* Addison-Wesley, 2005.

[60] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory To Implementation.* Morgan Kaufmann, 2^{nd} edition, July 2010.

[61] T. Porter and T. Duff. Compositing digital images. *SIGGRAPH Computer Graphics*, 18:253–259, Jan. 1984.

[62] M. Potmesil and I. Chakravarty. A lens and aperture camera model for synthetic image generation. In *Proceedings of the 8th Aannual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '81, pages 297–305, Dallas, Texas, USA, 1981. ACM.

[63] M. Potmesil and I. Chakravarty. Synthetic image generation with a lens and aperture camera model. *ACM Transactions on Graphics (TOG)*, 1 (2):85–108, Apr. 1982.

[64] F. Randima, editor. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics.* Pearson Education, 2004.

[65] A. Reshetov. Morphological antialiasing. In *Proceedings of the 2009 ACM Symposium on High Performance Graphics*, pages 109–116, New Orleans, Louisiana, USA, Aug. 2009. ACM.

[66] G. Riguer. Real-time depth of field simulation. In W. F. Engel, editor, *ShaderX*^{2}*: Shader Programming Tips and Tricks with DirectX 9*, pages 529–556. Wordware Publishing, Oct. 2003.

[67] P. Rokita. Generating depth-of-field effects in virtual reality applications. *IEEE Computer Graphics and Applications*, 16:18–21, Mar. 1996.

[68] T. Scheuermann and N. Tatarchuk. Improved depth of field rendering. In W. Engel, editor, *ShaderX*^{3}*: Advanced Rendering Techniques in DirectX and OpenGL*, pages 363–378. Charles River Media, 2004.

[69] C. Scofield. $2\frac{1}{2}$-D depth-of-field Simulation for Computer Animation. In D. Kirk, editor, *Graphics Gems III*, pages 36–39. Morgan Kaufmann, Jan. 1994.

[70] J. Selan. Using lookup tables to accelerate color transformations. In M. Pharr and R. Fernando, editors, *GPU Gems 2: Programming Techniques for High-performance Graphics and General-purpose Computation*, chapter 24, pages 381–392. Addison-Wesley, 2005. URL http://developer.nvidia.com/node/43.

[71] O. Shishkovtsov. Deferred shading in S.T.A.L.K.E.R. In M. Pharr and R. Fernando, editors, *GPU Gems 2: Programming Techniques for High-performance Graphics and General-purpose Computation*, chapter 9, pages 145–166. Addison-Wesley, 2005. URL http://developer.nvidia.com/node/27.

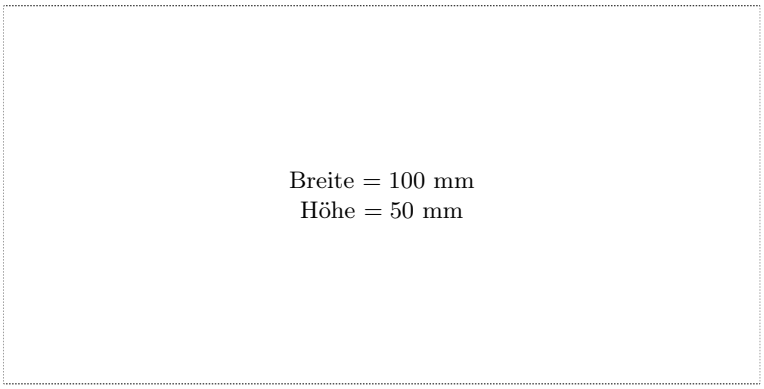
[72] M. Strengert, M. Kraus, and T. Ertl. Pyramid methods in GPU-based image processing. In *Workshop on Vision, Modelling, and Visualization*, VMV '06, pages 169–176, Aachen, Germany, 2006.

[73] L. Williams. Pyramidal parametrics. *SIGGRAPH Computer Graphics*, 17:1–11, July 1983.

[74] T. Zhou, J. X. Chen, and M. J. Pullen. Accurate depth of field simulation in real time. *Computer Graphics Forum*, 26(1):15–23, 2007.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —