FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Depth of Field

Point Splatting on Per-Pixel Layers

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Christoph Kollmann
Registration Number 1126633

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dr.techn. Dipl.-Mediensys.wiss. Przemyslaw Musialski

Vienna, 15th September, 2015

Christoph Kollmann

Michael Wimmer

Erklärung zur Verfassung der Arbeit

Christoph Kollmann

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. September 2015

Christoph Kollmann

Kurzfassung

Diese Bachelor Arbeit untersucht einen Depth of Field Rendering Algorithmus, welcher eine mehrschichtige Version von Point Splatting zur Realisierung verwendet. Die intuitive Prozedur verwendet ausschließlich modifizierbare Linsenparameter einer Kamera um den erwünschten Effekt in Echtzeit darzustellen. Nur anhand der Tiefeninformationen der Szene wird der Depth of Field Effekt berechnet.

Die so genannten Circles of Confusion stellen eine Einheit dar, um das Maß der Unschärfe für Objekte außerhalb des Fokuses zu repräsentieren. Abhängig von den Distanzen zwischen der Brennebene und dem momentan betrachteten Pixel wird einer Linse entsprechend eine Vergrößerung durchgeführt. Die anschließend benötigten Daten werden anhand einer Intensitätsverteilungsfunktion zur Verfügung gestellt.

Um Intensitätsverlust zu vermeiden und ein vollständig opakes Bild zu erhalten, wird der Effekt nicht kontinuierlich sondern diskret auf den echten Bildpunkten angewandt. Durch diese Einschränkung wird Subsampling vermieden und die Alpha-Wert Normalisierung ist nahezu perfekt.

Die Intensitätsverteilungsfunktion wird dem entsprechend ebenfalls in einer diskreten Form benötigt. Als Erweiterung zu diesem Algorithmus, wird ein vorlaufender Prozess zusätzlich verwendet, um die kontinuierliche Funktion zu diskretisieren. Das verwendete Shader Programm nutzt die Intensitätsverteilungsfunktion um auf einer hochauflösenden Textur diskrete Versionen zu zeichnen. Ein weiterer Beitrag präpariert diese Textur über Normalisierung, wodurch der benötigte Speicher steigt, aber der Rechenaufwand deutlich reduziert wird.

Der letzte Abschnitt des wissenschaftlichen Artikels inkludiert eine Defocused Highlights Implementierung, welche über eine Farbanpassung zur Imitierung verwendet. Generell wird die Helligkeit eines Pixels anhand einer einfachen Formel berechnet, bei welcher die einzelnen Farbkanäle unterschiedlich gewichtet betrachtet werden. Abhängig von diesem Wert und einstellbaren Parametern wird die Farbe in hellen Bereichen dem Effekt entsprechend angepasst.

Diese Bachelor Arbeit implementiert und erweitert diesen Algorithmus um die Leistung nach sieben Jahren zu beurteilen. Anhand unterschiedlicher Konfigurationen werden die erreichten Bildraten pro Sekunde als Metrik zur Bewertung und zum Vergleich zu früher genutzt.

Abstract

This bachelors thesis is about a depth of field rendering algorithm using point splatting on per-pixel layers. The intuitive procedure just uses specific adjustable parameters for the lens of a camera to generate the desired effect in real-time. Using nothing but the depth information of a scene, the depth of field can be computed.

The so called circles of confusion are the representing value of the blur rate for objects that are out of focus. Depending on the distance between the focal plane and the current pixel, an expansion takes place because of the redirecting nature of a lens. The following required information is retrieved using an intensity distribution function.

To avoid intensity leakage and provide a full opaque picture, the entire effect is done discrete on the screen space. This prevents subsampling and enables almost correct alpha value normalization.

As stated before, an intensity distribution function is required, but in a discrete form. This is an addition to the paper, where that function is preprocessed in the application and provided for the shader directly. A high resolution texture of the desired intensity distribution function is provided as input to a shader program that renders multiple discrete versions. Another further addition prepares the texture by normalization, consuming more memory, but saving lots of time during computation.

The last part of this paper includes a defocused highlights implementation, where a tone-mapping is applied to mimic them. In general, the luminance of a pixel is computed by a simple formula, summing up the three colors with adjustment factors. Depending on this value and some parameters, the color is adjusted where bright regions are enhanced to construct this effect.

This thesis implements and extends this work and is used to test its performance after seven years when it was published. Using different configurations for the effect, the provided stable framerates of the results are used as metric for benchmarking.

Contents

List of Figures

List of Algorithms

CHAPTER 1

Introduction

Computer graphics is one of many fields of computer science and is about artifical generation or composition of digital images. There are lots of subfields, focusing on different aspects like image processing or visualization. This thesis is working with fully artificial scenes, where only digital data is processed and transformed into pictures. That procedure is called rendering and is distinguished into an offline and online form.

The difference between those two separations is given by the required amount of time for termination. If the effort is done extremely fast, like thirty times per second, the classification is real-time rendering, also known as the online rendering form. When the required amount of time is much higher, it is called offline rendering or pre-rendering. This bachelors thesis topic deals with rendering in real-time, where an artificial digital scene, that is composed of several objects arranged in a virtual worldspace, is computed to a resulting image.

The demands of the image may differ from a picture close to reality or a comic like feeling. To make an artificial image appear real, different algorithms are used to illuminate the scene, where the light transport is simulated. Current hardware is capable of doing this on appropriate data, but when the requirements increase the time effort grows directly proportional. Rendering a set of different scaled cubes is lots faster than an entire complex city. Not only light transport is reconstructed in the rendering procedure, also typical effects for the human vision are mimiced. Examples for such occurences are motion blur and depth of field and may require high amounts of time.

1.1 Problem Statement

This work deals with the artificial simulation of depth of field in real-time applications. The challenge is to implement a new published idea to achieve good performance with a correct approximation of the physical reality. Depth of field stands for the effect of light transport through lenses, causing unfocused targets to be blurred. Since the human

eye consists of a lens, this effect is most natural for our perception. The reason why it is different for rendering to compose, lies in its nature as a pinhole camera. In the rendering procedure, there is no lens that redirects the light rays. Everything is perfectly sharp and there is no influence of the distance between object and camera.

To achieve this effect, the factor of blurring is computed using the *circles of confusion* [LKC09]. Basically the effect is reconstructed just with the distance difference using several formulas and decision metrics. Considering the circles of confusion using a lens model, there are a few adjustable parameters in a camera:

- lens size
- focal length
- focal depth

The lens size describes the actual diameter of the lens that manipulates the direction of light rays. The focal length represents the strength of the convergence or divergence of light. The focal depth defines the focal plane, at which distance the scene is sharp. Combining all these parameters, the actual size of every circle of confusion is determined by its depth value of rendered image. Refer to Figure 1.1 for a visualization of the three degrees of freedom.

Since it is dealing with calculations that may take lots of time, the aim is to reduce the computational effort to a minimum for a fluent and continuous visualization. The main focus lies on optimizations, things that might be cut off or approximated easier than other ways to save time.

Basically, the motivation of this bachelors thesis is about the implementation and expansion of a new way to mimic the performance costly depth of field effect in a real-time environment.

1.2 Goals and Contribution

The challenge to implement the depth of field effect is the computational effort. This bachelors thesis goals are separated into three single parts.

- Implementation of the depth of field rendering algorithm

The reimplementation the idea of the paper Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers [LKC09]. Having a working implementation of the algorithm, the second point takes place.

- Contribute to the working algorithm

Regarding possible optimizations for this effect it is important to make a decision between memory consumption and computational effort. In the last step the changed performance will be compared.

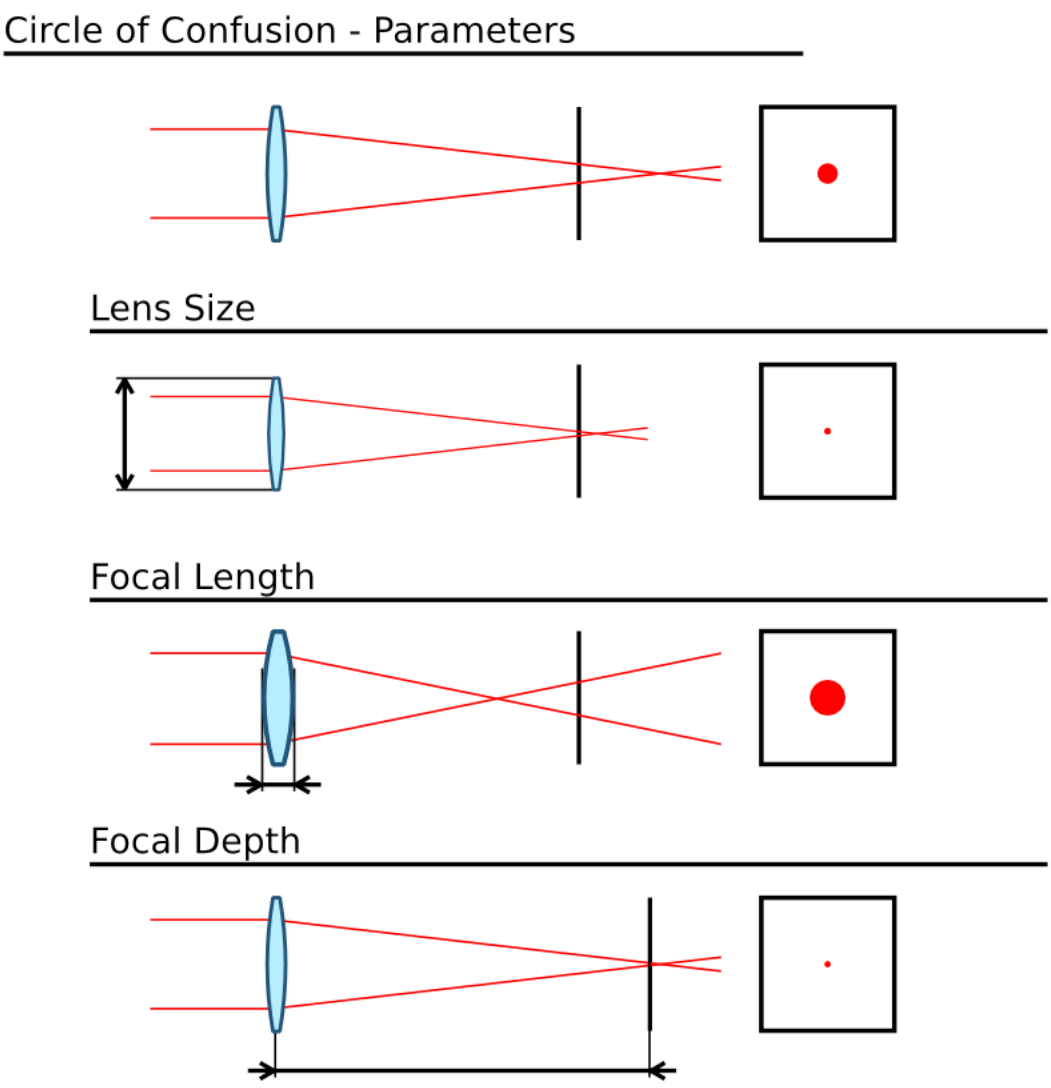

Figure 1.1: Illustration of the adjustable lens parameters.

- Benchmarking with own metrics

This is necessary to see the increased capabilities of todays hardeware, taking the results of seven years ago into consideration.

In general, this depth of field algorithm is implemented along with other components in a global illumination framework. Therefore its implementation must be designed in a way, that is not interfering with other graphical effects.

CHAPTER 2

State of the art

There are several different approaches to create the depth of field effect, mostly fighting against a few common artifacts. The separation into the offline and online rendering forms are significant regarding the rendering speed. In the offline rendering, also known as pre-rendering, the time of effort is not critical. It is used in rendering animated movies to create most realistic scenes, mostly with global illumination.

Regarding the online rendering, also known as real-time rendering, the required time for computation must be minimal. The single approaches for depth of field rendering can be distinguished into postfiltering attempts and in multipass rendering.

- Postfiltering

Postfiltering on one layer is known to be the fastest way to go, but is mostly unable to counter artifacts like intensity leakage and partial occlusion. In the simplest version, the source image is distributed to individual groups depending on their depth values. Afterwards, each pixel is treated individually by applying blurring techniques like fast fourier transformation, pyramidal filtering, anisotropic heat diffusion and gaussian blur [SW12]. The reason for group distribution lies in the necessarity of correct alpha blending. Using this simple postfiltering, artifacts like color bleeding to the background occurs where it is not physically correct.

The main issue applying depth of field rendering algorithms to single layered images is the partial occlusion that is not possible because background information is absent. Countering that issue leads to the next great group of approaches, where several layers are used to apply a correct blur. Procedures like depth peeling are necessary to distribute the entire scene to lots of layers to treat them individually. The natural problem that occurs are memory pressure and too many render calls that lead to bad performance. Partial occlusion is countered, but the effeciency is depending on the complexity of the scene to be rendered [SW12, LKC08].

The postfiltering approaches are separable into two main groups, the scatter methods and the gather methods. In the scatter methods, the information of each pixel is used

and speaded into the scene, extending these to sprites [LKC08]. Where lots of sprites for a large display have to be rendered, the performance drops very fast when the effect's intensity is increased. This means, the blur rate that depends on the depth value, known as circles of confusion diameter, is indirect proportional to the performance. The problem with lots of sprites drawn to an image plane is primarily the correct alpha blending, where all sprites have to be depth sorted very expensively.

Gather methods do not distribute pixels as sprites to the scene, they collect data around the pixel, taking a look at their neighborhood [KTB09]. As postfiltering in image processing states, it is like convoluting the image with custom kernels. For example, data is collected depending on the depth values to prevent color bleeding to the background, if something is too far in the front.

- Multipass rendering approaches

A multipass approach is for example the accumulation buffer [HA90]. The accumulation buffer, as its name states, accumulates data from different render steps into one frame. To achieve the depth of field effect, a circular pattern is followed by the camera that always focuses the same point. Lots of samples are required to get satisfying results, resulting in bad computation time, that is not practicable in real-time. Another problem with this procedure lies in its nature where general lens models are not able to be mimiced. Distributed ray tracing procedures are another example for multipass approaches, where lots of samples are used to collect correct data. Both techniques are useful, because most all artifacts are not even encountered. Increasing the effect's intensity leads to greater circle of confusion sizes, where the rendering algorithms' load increases enormous.

Depth of field with nonlinear mipmap interpolation [LKC09] is countering artifacts like bilinear magnification artifacts, intensity leakage, and blurring discontinuity. This procedure uses anisotropic mipmaps with custom sampling, that collects the blur data on a circular style over the single mipmaps. The selection of the correct mipmap level is done using the standard deviations of a gaussian filter. For correct gather procedures, the degree of blurring and the degree of neighbors blur is computed individually.

As another multipass method, Realistic Lens Blur [LES10] approaches the depth of field rendering algorithm with a customized depth peeling. Since this effect is utilizing ray tracing, the amount of layers must be reduced for high performance. The ray refraction is done using two height maps to correctly adjust the rays into the scene. With several optimizations, like using minimum and maximum depth areas to clamp down rays, a fast intersection procedure, and neighborhood approximations for circle of confusion sizes the rendering algorithm achieves good render performances. For satistfying results, lots of rays are required and the amount of layers needs to be increased. Both variable values are requiring either computation time or memory consumption and are therefore just for limited use practicable.

Rendering the depth of field effect with dynamic light field generation [YWY10] uses some samples taken from different angles just like explained in the accumulation buffer [HA90]. The difference to the main image is greater and are used to sample down using the light field. Simulation of the depth of field effect is done using the spatial

integration and the angular information. There are already several approaches using this technique, with small changes like using the fourier space instead. The results are depending on the amount of sampled views, but provide very good performance when the additional memory cost is accepted. Problematic in this implementation are the boundary artifacts that are not countered completely.

Concluding from these different depth of field approaches, the rendering method must be able to counter all occuring artifacts that come along with the simulation procedure. As already mentioned, several layers are necessary to avoid the most common problem of partial occluded areas [SW12, LKC08]. Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers by Lee, Kim and Choi of 2008 is countering all these common artifacts [LKC08] as one of the postfiltering attempts. The only problem refers to the limitations of current graphical processing units. Since the approach is from 2008, the hardware capabilites have been increased steadily. Therefore, to evaluate the point splatting technique, and because of the resistance to common artifacts, this rendering algorithm is chosen. The contribution to this procedure basically balances the computational load to additional memory consumption, resulting in faster execution times.

CHAPTER 3

Method

This bachelors thesis is focusing on implementing the scientific paper Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers by Lee, Kim and Choi of 2008 [LKC08]. As already mentioned in Section 2, the reasioning for this choice takes the advantage of current hardware to further evaluate the rendering algorithms usability.

There are several single steps necessary to implement their idea for this effect. In essence, the Depth-of-Field rendering algorithm depends primarily on the expansion of single dots taken their depth value into consideration. As every pixel is transformed on its own, it is mapped to a larger sprite depending on their circle of confusion discussed in Section 3.2 [PC82]. The circle of confusion is, taking apertures of camera lenses into consideration, a directly describing transformation in size of unfocused objects. This leads to a blur effect on everything thats not in the plane of focus. Using this expansion, an intensity distribution function is required as mapping from a pixel to a sprite. Depending on the aperture of a camera, different shapes as shown in Figure 3.1 are constructed. The intensity distribution function is simply a texture of the desired Bokeh effect discussed later. This bachelors thesis also contains additions for performance gain and additional features, as Bokeh-Preprocessing described in Section 3.1.

Next to the rendering algorithm, several necessary steps are included to counter common artifacts as stated in Section 2. The rendered sprites are distributed to several layers for a correct blending regarding depth values. Separation itself is depending on the depth value using formulas just like the circle of confusions. This is called Point Splatting on Per-Pixel Layers, that is the title of the paper this work bases on. The explanation for this technology is described in Section 3.3. Distributing the rendering algorithm to several layers, an additional merge pass is required to compose the single information sources into a final image in Section 3.4. As a last feature, the implementation of defocused highlights is included using a color adjustment technology. The used tonemapping is mimicing them with extrapolating the light intensity values, explained in Section 3.5 [Lan02].

3.1 Bokeh-Preprocessing

The term Bokeh has its origin in the japanese language and stands for unfocused or blurred. Its commonly known and used in photography to describe individual aperture patterns for camera lenses. For the depth of field occuring in the human eye, no special aperture is required since its just a circular lens. This results in a circular blur for parts of the image that are not in focus. Special patterns are constructed by blocking the path of light input right before the lens as shown in Figure 3.1.

Figure 3.1: Left: Custom aperture self created to produce heart shaped bokeh effects. Right: Application of the custom aperture shown left. Online 09.11.2014: http://www.diyphotography.net/diy_create_your_own_bokeh

To support these patterns, the intensity distribution function must be supplied for the mapping procedure. As addition to the work and the framework, this project is able to convert any image as a texture into desired function. Applying the texture directly as sprite expansion, there is a problem within regarding normalization issues. Taking lots of overlapping sprites with different color and alpha values, the merging of these will lead to "hyper" and "hypo" intensities [MvL00]. This is countered later in the merging step, but is because of subsampling and scaling factors not easy to handle.

The result of applying the intensity distribution function directly will lead to rounding errors and intensity leakage. Since the Depth-of-Field Point-Splatting is applied on screen space only, the Bokeh pattern is also applied discretically. Therefore, the normalization of pixels extended to the same value are not required to be normalized. But the alpha values will still fluctuate due to different extensions next each other, so the normalization error is present in small extent. Since every pixel is swapped with an expanded texture, the dimensions must increment in the matter of natural numbers.

Providing an image of the desired pattern, our framework initiates a computation to generate the required texture strip. Due to performance issues, very large scaled circles

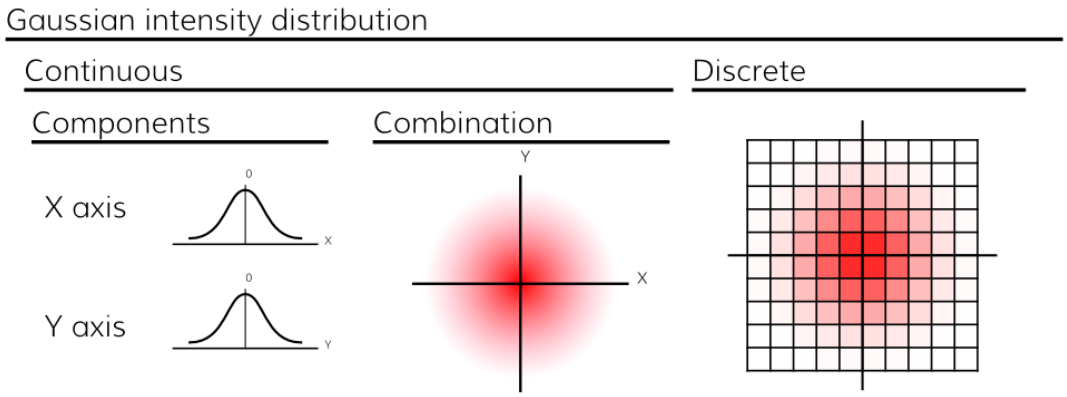

Figure 3.2: Discretization of a gaussian intensity distribution. Both axis are combined and depending on the distance to the center in natural numbers increased and averaged.

of confusion are not practicable and our procedure limits to the maximum diameter of 64 pixels width and height. The further details on its implementation is located in section 4.1

3.2 Circles of Confusion

The Circles of Confusion are a metric to describe the extent of pixel resizing to achieve the blurring effect. A point that is out of focus is directly transformed into a picture of the intensity distribution functions texture strip and expanded to match the required size. Afterwards the resulting sprite is pasted onto screen coordinates because of its discrete nature described in Section 3.1.

This effect is natural to humans, since the eye collects the incoming light information through a lens with the associated redirection of the single rays. The already described intensity distribution function is a two dimensional function to map distances with varying intensity values as shown in Equation 3.1[BW02].

$$PixelOpacity = f(x, y) \tag{3.1}$$

Intensity distribution functions can be anything imaginable, for example a distribution indirectly proportional to the distance to the center like shown left in Figure 3.2. To reduce the amount of calculation calls, and because of the nature of the discrete step lookup, the function can be reduced to a two dimensional texture as shown right in Figure 3.2. This saves lots of computing time and reduces the demand to only a few texture lookups. Taking gaussian or union distributions into consideration, this might be a overhead, but the effort does not increase with the complexity of the intensity distribution function.

The task to provide a intensity consistent image demands, that every pixel is transformed into a sprite with an overall opacity of one. In the paper, they use discrete pixel

expansion using the DirectX9 feature texturing extension of point sprites. To normalize the entire region of a single discretization, a parameter is chosen and inserted manually. As explained in Section 3.1, in this work this is done automatically with the texture strip generation and does not require additional normalizations during the procedure.

With the upcoming formulas in this section, the circle of confusion diameter is computed and ready for use in the desired effect [PC82].

Equation 3.2 uses the depth values from the current point of interest and the camera configuration parameters to compute the extend of the blur.

$$C_r = |W_z - W_f| \cdot \frac{E}{W_z} \tag{3.2}$$

C_r represents the circle of confusion diameter. E describes the lens size. The parameters W_z and W_f are factors determined by the lens properties and the depth values in Equation 3.3.

$$W_z = \frac{F \cdot Z}{Z - F} \quad \text{and} \quad W_f = \frac{F \cdot Z_f}{Z_f - F} \tag{3.3}$$

Combining the previous formulas, the substitution is shown in Equation 3.4.

$$C_r = |\frac{F \cdot Z}{Z - F} - \frac{F \cdot Z_f}{Z_f - F}| \cdot \frac{E}{W_z} \tag{3.4}$$

Some factors in this equation are only depending on the lens parameters and can be combined to one constant value. This must not be determined for each sprite itself and can be considered as great performance gain if computed once on parameter change on the central processing unit. Applying this optimization, the formula can be reduced by filtering out the constant terms like shown in Equation 3.5

$$C_r = \frac{E \cdot F}{Z_f - F} \cdot \frac{|Z - Z_f|}{Z} \tag{3.5}$$

As annotation, the single parameters can be simplified to simple values that are present during the rendering algorithm shown in Equation 3.6.

$$\text{CoC-Index} = \mathit{Constant} \cdot \frac{|\text{Depth} - \text{FocalDepth}|}{\text{Depth}} \tag{3.6}$$

The *Constant* in the previous equation is depending on the lens configuration only as in Equation 3.7.

$$\text{Constant} = \frac{\text{FocalLength} \cdot \text{LensSize}}{\text{FocalDepth} - \text{FocalLength}} \tag{3.7}$$

When the circle of confusion diameter is computed, the sprite expansion takes place. To recreate the feature removed in DirectX10, a shader program is required with a vertex shader, geometry shader and a pixel shader. The raw data to be processed is a list of vertices for each pixel. For a resolution of 800x600, there are 480.000 vertices input for the shader program. Their content is limited to just the pixels position in natural numbers, further information like the color data and depth value are read back from the source image.

Implementation details are described in section 4.2.

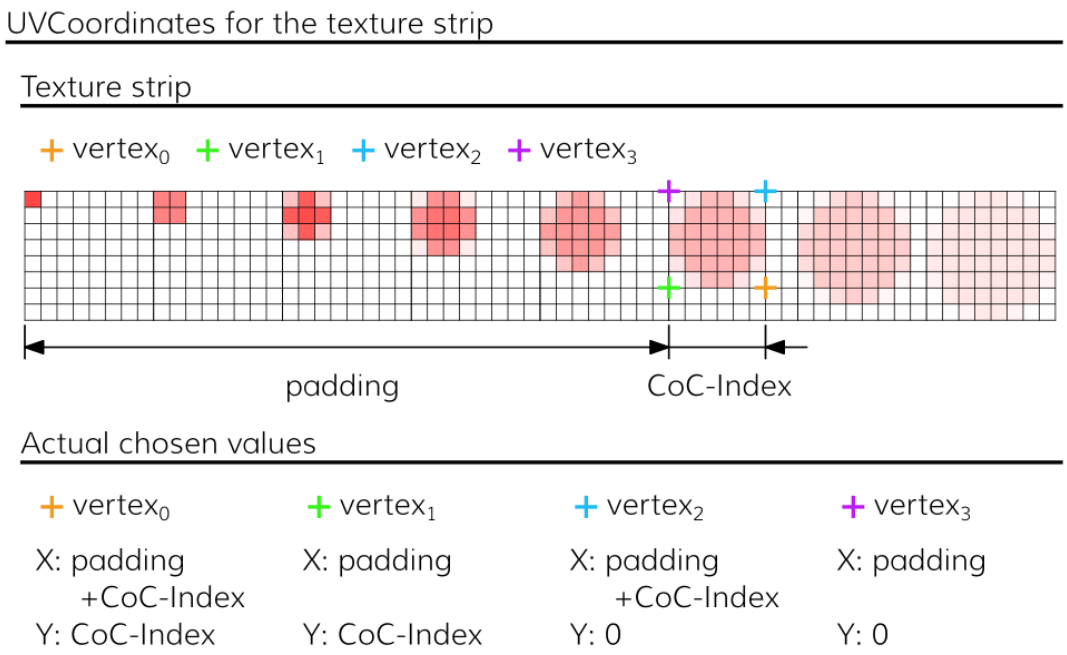

Figure 3.3: Illustration of the UVCoordinates selection as preparation for texture lookup calls. CoC-Index stands for the index of the present discrete versions of the intensity distribution function as well as its circle of confusion diameter.

3.3 Point Splatting

The Point Splatting technique uses the preprocessed bokeh texture strip from Section 3.1 and pastes them directly on screen space. Using this transformation, a pixel is expanded into a sprite with the resolution chosen by the circle of confusion diameter in Section 3.2 and blurs the image for the depth of field algorithm.

Since every pixel is splatted individually, the resulting image is physically correct without intensity leakage. With the circle of confusion calculation, seen in the geometry shader in Algorithms 4.1, 4.2 and 4.3, the next step is the pixel shader. The four new vertices contain the following data:

- Color in RGBA-Components (Red Green Blue Alpha).
- UVCoordinates on the texture strip.
- Depth value.

The color data is retrieved from the previous render procedure as well as the depth buffer. UVCoordinates are representing the selection of the computed circle of confusion diameter as shown in Figure 3.3. The CoC-Index value stands for the circle of confusion diameter that also represents the index of the tile in the bokeh pattern texture strip

generated in Section 3.2. To counter the problem of missing information behind the objects that are blurred, the second layer behind the very first visible objects is also taken into consideration. This operation is applied twice, for the first and for the second layer depending on their depth values. In the refering paper implementing this technology [LKC08], there are a few selection criterias for the second layer as optimization.

First of all, a small threshold is used to prevent rendering the same object on the layer behind the first one. This might lead to artifacts like wrong color intensities as some points will receive twice as much opacity.

Second, all data in the background is skipped. This is comparable with the first optimization step, but has its main focus on the unnecessary information and performance loss. The background is most likely to be overall blurred at the same extend without focused or unfocused object behind it. If the depth buffer reaches a certain value, a pixel is considered as background and for the second layer dropped.

And the last one is to only consider pixels that will be actually required by the effect. This means, that only the partially occluded areas pixel information behind the blurred object is required. The implementation in the paper [LKC08] uses a neighbor analyzation to check if there are great changes in depth values for pixels next each other. In post processing technologies this is comparable with edge detectors that check the gradient of an image. Computation for this threshold is done using samples on the neighborhood with a kernel.

Since there is a issue with partially occluded areas, a correct alpha blending is required for satisfying results. This is where the Point Splatting technology takes place, as the new data must be distributed for a correct blending. There are three possibilities for the destination of a new pixel: above the source, on the same level, and behind.

The selection model for this distribution is illustrated in Figure 3.4.
Depending on Equation 3.8, the correct layer is selected and used as render target to benefit from the built in alpha blending of DirectX.

$$\text{Layer} = \begin{cases} 0 & \text{if } B < C_1 \\ 1 & \text{if } C_1 \leq B \leq C_2 \\ 2 & \text{if } C_2 < B \end{cases} \tag{3.8}$$

The variable B is the evaluation criteria for each pixels depth value and defined in Equation 3.9.

$$B = B(p) - B_v(q) \tag{3.9}$$

Where $B(p)$ refers to the depth value of the currently splatted pixel and $B_v(q)$ to the depth value of the already present destination pixel. The depth value is used in a signed version of the calculation for the circle of confusion diameter as shown in Equation 3.10.

$$\text{CoC-Signed} = \text{Constant} \cdot \frac{\text{Depth} - \text{FocalDepth}}{\text{Depth}} \tag{3.10}$$

The value of the Constant variable is defined in Equation 3.7. The borders C_1 and C_2 are also computed by using the depth value of the already present pixels depth information,

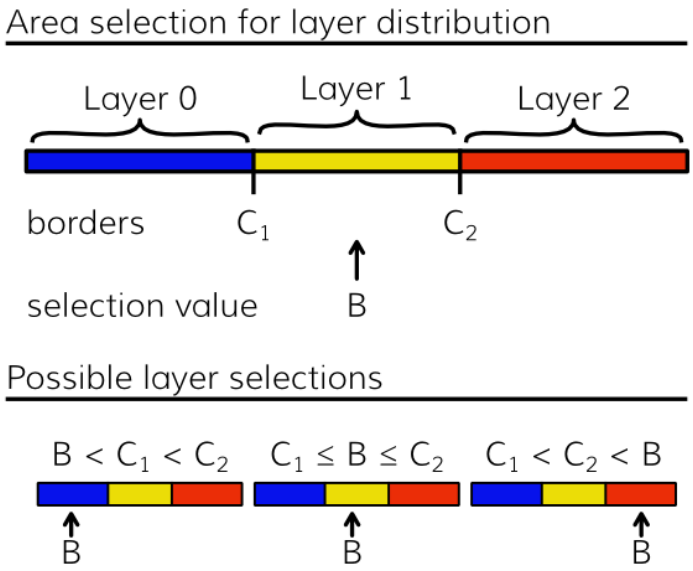

Figure 3.4: Selection model for layer distribution depending on a selection variable B and two defined borders C_1 and C_2.

but with the sign removed. An additional tolerance parameter is also applied to the borders to enlarge the group of all pixels on the same level. The application of this layer selection and the resulting layer distribution is illustrated in Figure 3.5.

As the figure shows, only one pixel is taken into consideration at a time. For every new destination of the sprites location, the depth value of the vertices is taken and compared with the source image that is being transformed. The circle of confusion diameter is computed individually and then the distribution takes place using the provided selection parameters. Simplified, pixels above the source pixels are assigned to layer 0, the pixels on the same level as the source image are in layer 1 and the ones behind the source are assigned to layer 2. This separation is required to achieve better results as post-processing depth of field implementations, because each pixel is treated individually and separated for enhanced alpha blending.

The reason for this being run twice, for the first and the second layer, is illustrated in Figure 3.6. Assuming the circle of confusion diameter computation results to seven, then a edge pixel is speading into other objects by three pixels and four pixels are not covered with background information. The three pixels will be correctly blended over the source image as the layer distribution is done correctly regarding present depth values. But the other four pixels, that move into the object being blended, will experience great changes in opacity and lead to a transparent occurance that reveals the background information. If there is no second layer behind there regions, then it will be blended with the clear color of the renderer and result in unacceptable artifacts.

The alpha blending is described in the next Section 3.4 when the individual layers are blended together for the resulting image. An additional normalization is required to

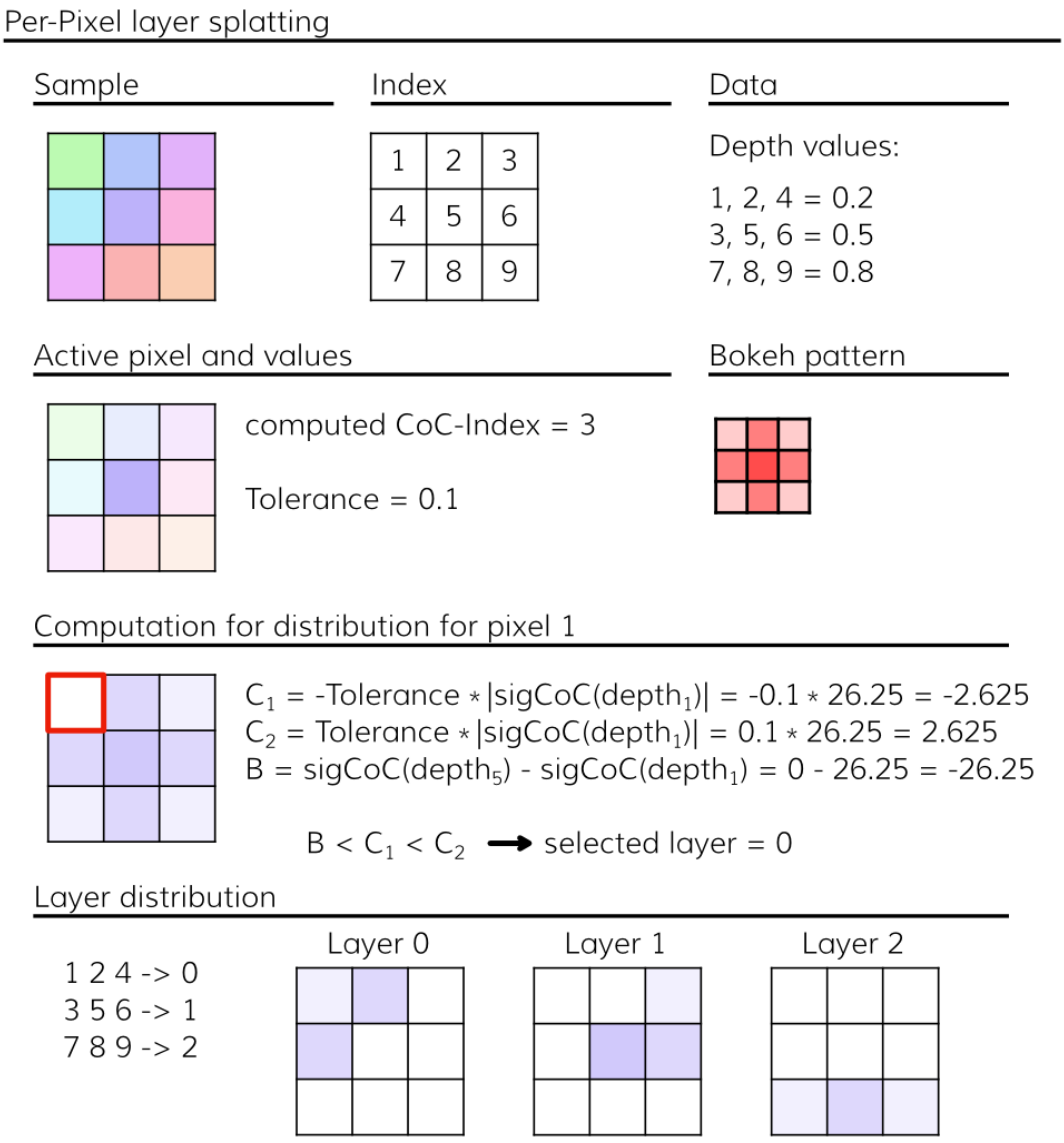

Figure 3.5: Per-Pixel layer splatting illustration. The center pixel is splatted as sprite on the 9 fields and is distributed by computing the values for the decision.

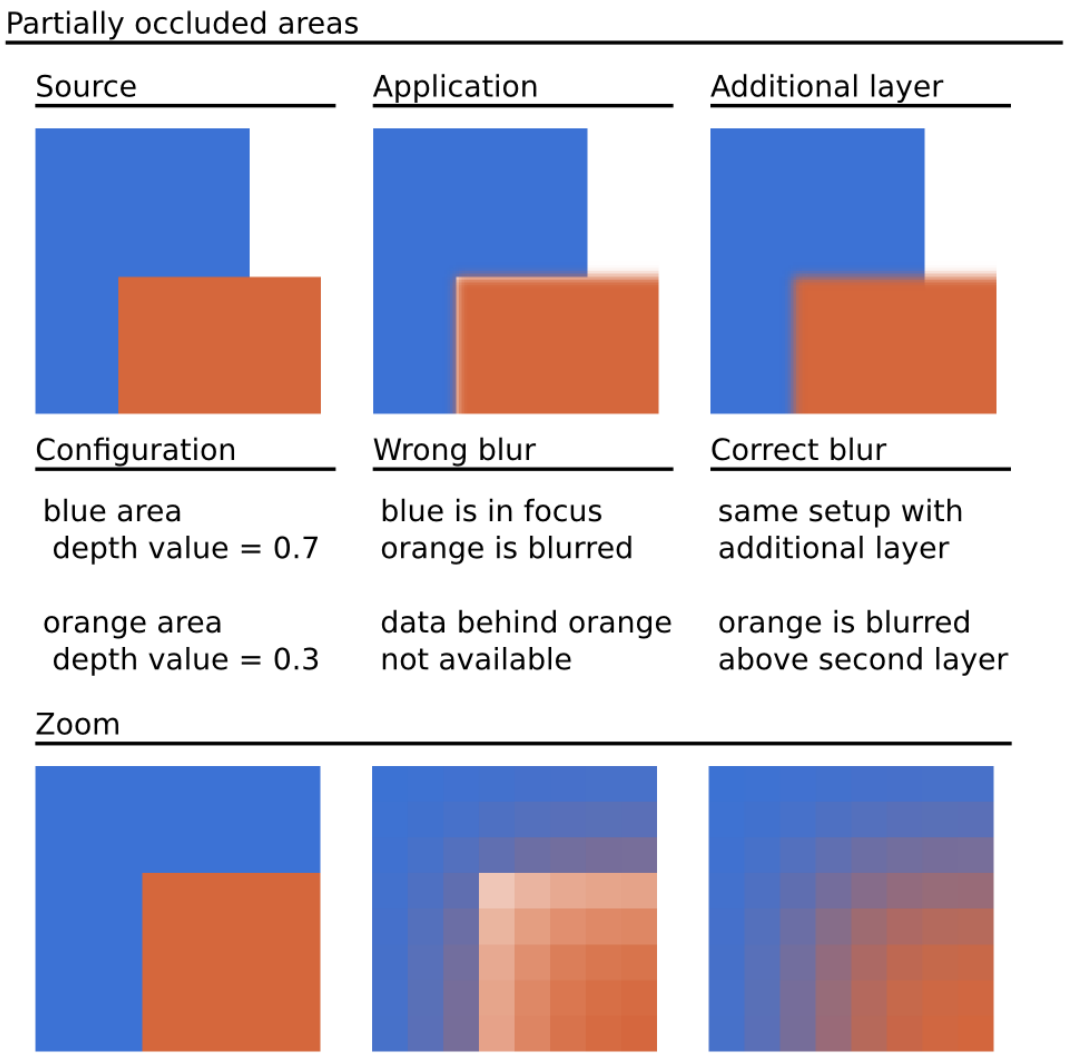

Figure 3.6: Challenge: partially occluded areas. The source shows a configuration with two quads that have different depth values. The middle graphics are applying the effect on screen space and the right graphics are enhanced with an additional second layer.

correct the alpha intensities that occur in a minimal manner, since discrete and already normalized intensity distribution functions are used.

3.4 Merging and Normalization

In the merging procedure, there are overall five layers to be merged down to one final result image. The first layer, the one most in the front, is distributed to three layers

according to above, at the same level, and below like described in Section 3.3. These are called L_0, L_1, and L_2 for above, same, and behind. The second foremost scene information is distributed to only two layers, because there is no data required for elements that are above, since its in the hidden region [LKC08]. These are called M_1 and M_2 for same and behind.

The same and behind layers are simply summed up since its in the same configuration and not required to be blended over each other. For the correct ordered alpha blending, the data is already correctly sorted and can be composed from back to above [BBGK02]. The last layer is used and multiplied with the next layers inverse alpha values. Then, the result is summed up with the layer where the inverse alpha values are taken from. This is continued until all layers are summed up. Equations 3.11, 3.12, and 3.13 represent this alpha blending procedure.

$$result = L_2 \tag{3.11}$$

$$result = L_1 + (1 - L_1.alpha) \cdot result \tag{3.12}$$

$$result = L_0 + (1 - L_0.alpha) \cdot result \tag{3.13}$$

After each addition, the *result* is normalized by the division for each pixel by its own alpha value. This is required to counter the leakage as well as the values above one.

3.5 Defocused Highlights

The defocused highlights implementation is an addition to the depth of field render algorithm in the refering paper [LKC08]. It is more or less just a color adjustment as a post-processing step after the rendering procedure is done. This is called tone-mapping and is used for color adjustments to enhance images. For example, a picture may be adjusted with several steps starting with white balancing. Then the histogram of the image's light intensities is stretched to maximize the contrast. In the last step, a clamping is done to remove artifacts appearing because of the adjustment.

This implementation is done in two steps:

- Compute the luminance for each pixel individually.
- Adjust the color value according to the computed luminance.

Depending on the red, green and blue components, the luminance is computed using the formula shown in Equation 3.14 [Lan02].

$$luminance = 0.3 \cdot Red + 0.59 \cdot Green + 0.11 \cdot Blue \tag{3.14}$$

Using this value, the values are adjusted using chosen thresholds and parameters that are free to choose. This adjustment is just like the depth of field rendering algorithm, completely independent on the way the source image is constructed.

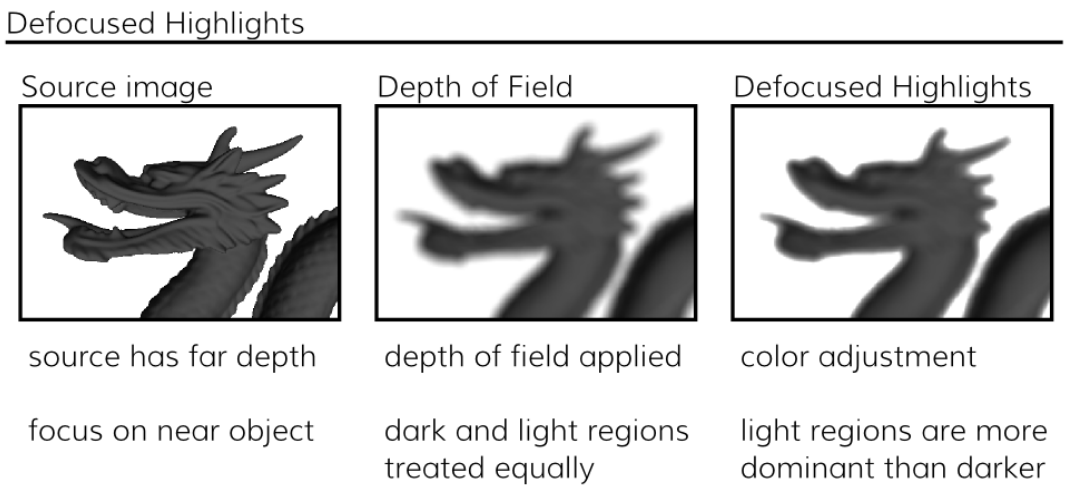

Figure 3.7: Illustration of the defocused highlights application.

3.6 Optimization: Quarter reduction

The depth of field rendering algorithm bases on blurring an image according to the respective depth values of the current pixel. To improve the rendering performance, the quality of the result stays the same with a simple trick. Since the blur rate is more or less linear blending, the resolution may be reduced and then subsampled overlayed.

This takes place for all circle of confusion diameters above three, where the resolution is halfed as well as the sprite expansion. The performance bottleneck lies within the size of the sprites and the resultion of the destination. This is directly interchangeable with the amount of pixel shader calls, that depends on these two variables.

Therefore, lots of pixel shader calls are saved if the resolution is halfed, resulting in only fourth as much pixels. Rendering that way saves computational time but increases the memory consumption by doubling the required layers. For each layer on full resolution, another with quarter resolution is present. The rendering takes place in sequence, in the first step for the computed circle of confusion diameters of smaller than four and then with the ones above three.

Before the actual merging described in Section 3.4 is applied, the color data of the quarter reduced layers is added in subsampling procedure to the full resolution layers. Refer to Section 5.1 about how much performance gain is achieved using this optimization.

CHAPTER 4

Implementation Details

The actual implementation notes of this bachelors thesis is discussed in this chapter.

4.1 Bokeh-Texturestrip Generation

A wide texture of dimensions 4096x64 is set as render target and with a padding of 64 pixels every size from 1 to 64 is drawn. As shown in Figure 4.1, a pattern is converted into texture strips for later use.

The generation contains several steps as preprocessing:

- Draw the texture with incrementing size.
- Count the total opacity of every intensity distribution texture.
- Normalize the alpha values.

The first part creates a new texture strip for storage of all 64 discrete versions of the Bokeh-pattern. To accomplish this render procedure, the paper discussed used the texturing extention of GPU point sprites of DirectX9. Since this is no longer present in the current specification, the procedure must be applied manually. This is done using a TriangleStream to convert the pixels to four vertices for expansion. Implementation of the geometry shader is discussed in Section 3.2. The render pass consists of a vertex shader, a geometry shader and a pixel shader. While the first one is trivial, just forwarding the vertex location, the geometry shader expands the dots to a triangle strip forming a quad. Every point is converted into four new vertices and the position is computed to fit discrete pixels. Depending on the index of the vertex, a new quad of the strip is used and expanded from the upperleft pixel.

In the paper this effect bases on, the normalization is done using an additional texture storing the values for alpha normalization for each sprite. The reason behind this is the intensity leakage problem of other Depth-of-Field approaches described in Section 2.

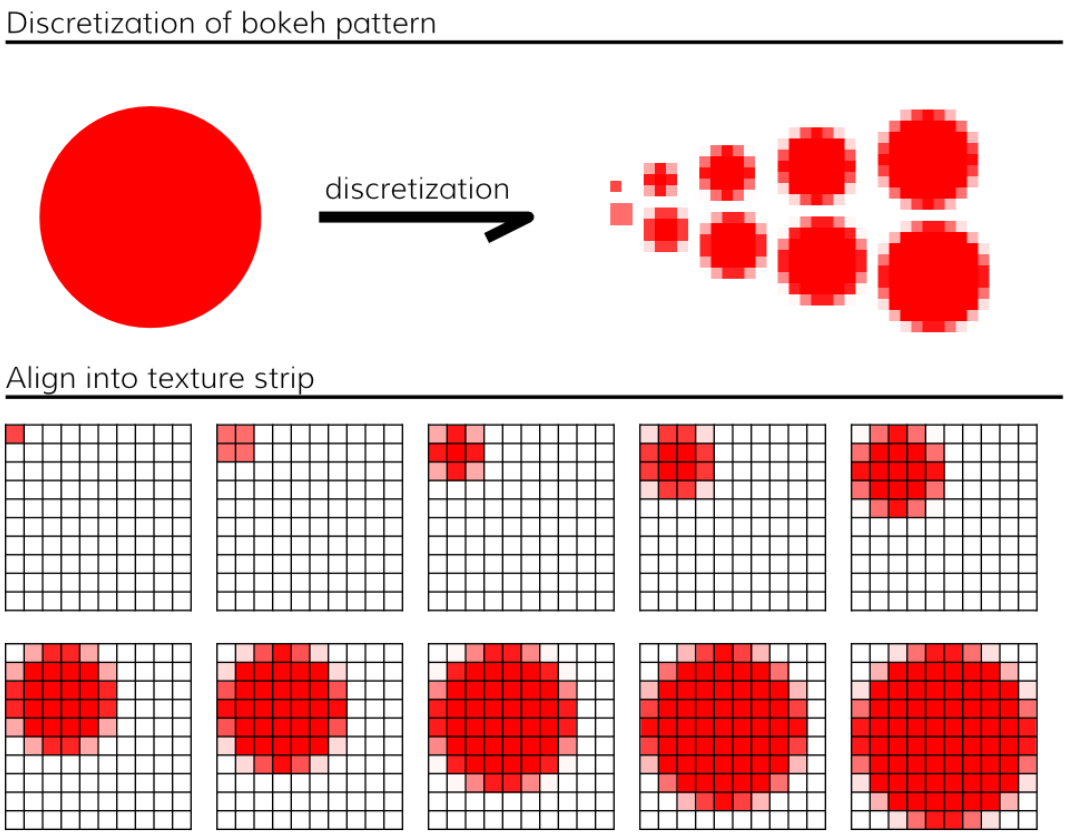

Figure 4.1: Rendering each discrete version of the bokeh-texture increasing size by one.

Basically, if the circle of confusion is eight, the expanded point covers the squared amount of pixels to be drawn. Therefore, taking the normalization into consideration, 64 divisions are necessary for one point to accomplish this step. Our idea increases precision on the bokeh texture strip with additional cost in memory consumption, but applies the alpha correction as a preprocess. This improves rendering performance by reducing the arithmetic functions required for each pixel.

Refer to Figure 4.2 as a full representation of the implementation.

An example: an intensity distribution function with a circle of confusion diameter of 2 units is mapped to a 2x2 pixel field. If the bokeh effect is a simple circle, the distribution will be symmetrical for each pixel. The rendering procedure of DirectX draws an anti-aliased version of the circle, so the alpha values will be depending on the scaling factor. Assuming the transparency value is 0.75 on each quadrant, the sum of all four pixels is 3. To achieve consistent opacity without "hyper" and "hypo" intensities, a points bokeh texture should sum up to 1 [MvL00].

The second pass is used to count all alpha values of every 64x64 section of the texture

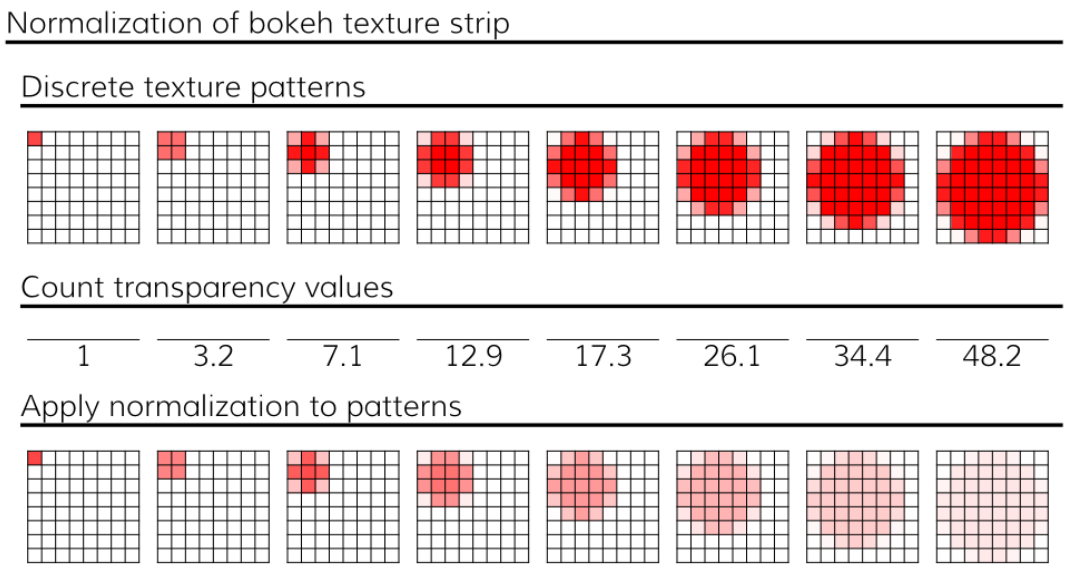

Figure 4.2: Representation of the texture strip generation. Top: discretization for every circle of confusion size; Middle: the sum of all alpha values for each discretization field; Bottom: the discretizaion divided by the counted alpha values;

strip. In the last pass, every pixel is divided by the resulting value for each section to normalize the texture strip. Therefore no normalization operations are required and the texture from the bokeh strip can be pasted directly onto screen space coordinates, since the discrete version does not require any subsampling steps.

4.2 Circle of Confusion - Shader Program

As described in section 3.2, the circle of confusion calculation is done using a complex shader program. It consists of a Geometry Shader, a Vertex Shader and a Pixel Shader.

The vertex shader is primitive and only forwards the coordinates to the geometry shader. In this step, the actual conversion from one vertex to four, forming a quad, is performed and position calculations are used to align the sprite correctly along the pixel grid. Figure 4.3 shows the task of the geometry shader, where CoC-Index stands for the circle of confusion diameter and respective for the position in the texture strip. The initial position information is used to identify the center point and is split into the four vertices mentioned.
With the depth information read back from the depth buffer, the circle of confusion diameter is computed for every pixel individually. In the geometry shader, discrete steps are precalculated for correct alignment. Since the rendering procedure is done in screen space only, and the intensity distribution function is used discrete, the point splatting must be aligned correctly for direct paste.

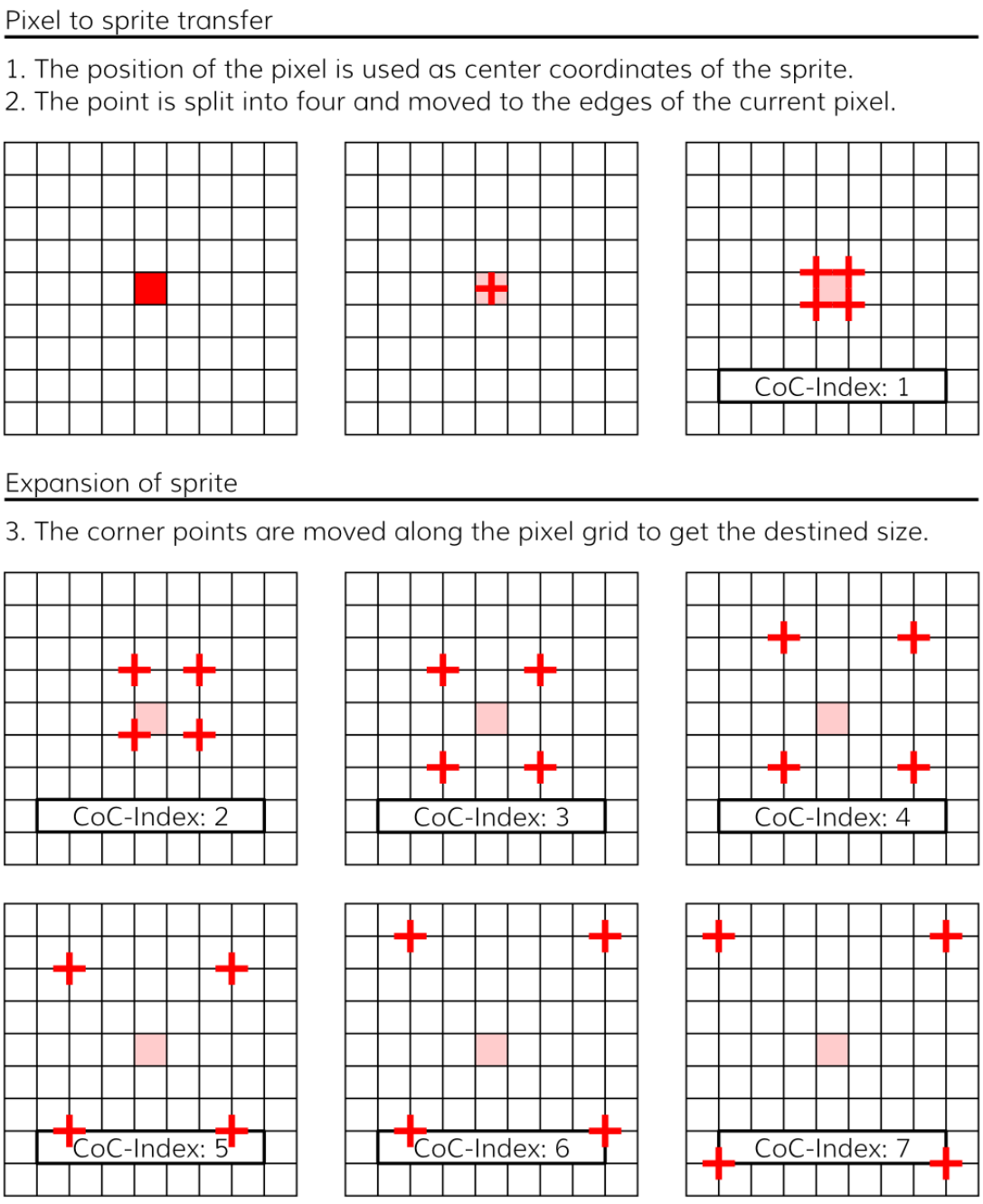

Figure 4.3: Illustration of the conversion from pixel coordinates to an expanded sprite.

The reasoning behind this requirement is to prevent subsampling and counter intensity leakage [LKC08]. This would be visible as opacity varying artifacts, resulting in high frequency noise in the image rendered. With a TriangleStream the vertices are moved and the texture coordinates are computed as shown in Algorithms 4.1, 4.2 and 4.3. The UV from UVCoordinates stand for the X and Y axes on the texture for data lookup, but is UV instead of XY because XY are already used variable names.

Algorithm 4.1: Geometry Shader - Preparation: The padding values and the grid aligned movements are precomputed for sum calculation.

Input: ColorData, DepthValue, PixelPosition

```
1 CoC = CoC(DepthValue);
2 padding = 64 * (CoC - 1);
3 extend = cocRadius / 2;
4 modulo = cocRadius mod 2;
5 stepX = 2 / (float)screenWidth;
6 stepY = 2 / (float)screenHeight;
7 extendPlusModulo = extend + modulo;
```

Algorithm 4.2: Geometry Shader - Position Determination: The destination on the pixel grid is computed.

```
1 movePositiveX = x + extendPlusModulo * stepX;
2 moveNegativeX = x - extend * stepX;
3 movePositiveY = y + extendPlusModulo * stepY;
4 moveNegativeY = y - extend * stepY;
5 paddingPlusCocRadius = padding + cocRadius;
```

In Section 3.3, the pixel shader is discussed that performs after the geometry shader finished.

4.3 Point Splatting Sprite Transformation

Depending on this number, the correct selection of pixels from the texture is read back and pasted on the resulting image like shown in Figure 4.4.
In the shader, this distribution is done with two simple if statements like shown in Algorithm 4.4.

4.4 Defocused Highlights Shader

The shader used for the defocused highlights is done entirely in the pixel shader stage. Algorithm 4.5 shows the implementation, where at first the luminance is determined, then modified and then applied to the current color value.

Algorithm 4.3: Geometry Shader - Texturestrip output: The output is computed by streaming out the new four vertices with given data about color, depth, position and UVcoordinates.

Output: Coordinates and UVCoordinates for four vertices

```
1  output.color = ColorData;
2  output.position = float4(movePositiveX, movePositiveY, z, 1);
3  output.texCoord = float2(paddingPlusCocRadius, cocRadius);
4  outStream.Append(output);
5  output.position = float4(moveNegativeX, movePositiveY, z, 1);
6  output.texCoord = float2(padding, cocRadius);
7  outStream.Append(output);
8  output.position = float4(movePositiveX, moveNegativeY, z, 1);
9  output.texCoord = float2(paddingPlusCocRadius, 0);
10 outStream.Append(output);
11 output.position = float4(moveNegativeX, moveNegativeY, z, 1);
12 output.texCoord = float2(padding, 0);
13 outStream.Append(output);
14 outStream.RestartStrip();
```

Algorithm 4.4: Layer Distribution: The comparation parameters are computed and prepared for two if statements for selecting the correct render target.

Input: ColorData, Depth, SourceDepth, toleranceDOF

Output: Render Targets Layer 1,2 and 3

```
1  sigDepth = sigCoC(SourceDepth);
2  distr = sigCoC(Depth) - sigDepth;
3  comp = toleranceDOF * abs(sigDepth);
4  if distr < -comp then
5  |  output.colorTarget0 = ColorData;
6  else
7  |  if distr <= comp then
8  |  |  output.colorTarget1 = ColorData;
9  |  else
10 |  |  output.colorTarget2 = ColorData;
11 |  end
12 end
```

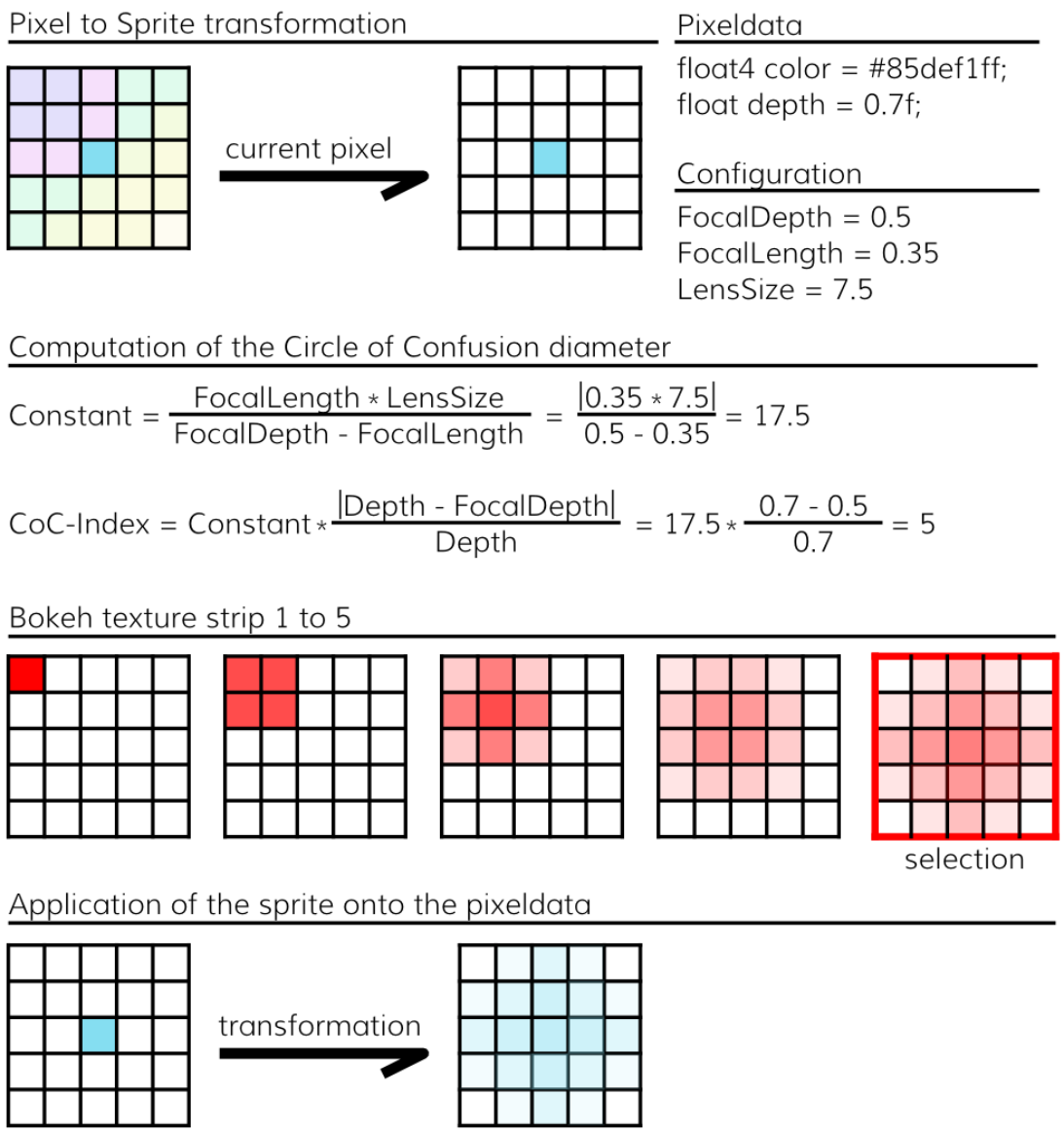

Figure 4.4: Illustration of the transfer of one pixel with the given parameters to the selected sprite in the bokeh texture strip. The constant is determined once upon change in configuration and the CoC-Index is computed on demand for each pixel.

Algorithm 4.5: Defocused highlights: The luminance is calculated and modified depending on a threshold, beta as exponent for falloff and gamma as expanding gain.

Input: ColorData, threshold, beta, gamma
Output: Adjusted ColorData

```
1 luminance = 0.3 * ColorData.r + 0.59 * ColorData.g + 0.11 * ColorData.b;
2 if luminance > threshold then
3 |   factor = (luminance - threshold) / (1 - threshold);
4 |   value = pow(factor, beta);
5 |   ColorData.rgb = ColorData.rgb * ((1 - value) + value * gamma);
6 end
```

CHAPTER 5

Results

This idea to create the depth of field effect is a valid procedure for a natural approximation. While it attemps to counter lots of issues seen in other ideas, the main issue lies on the real-time aspect. Its paper was released 2008, where the effect was only capable with the circle of confusion diameter up to an average of eight in real-time. Figure 5.1 shows the amount of required pixel shader calls for two resolutions depending on the average sprite sizes.

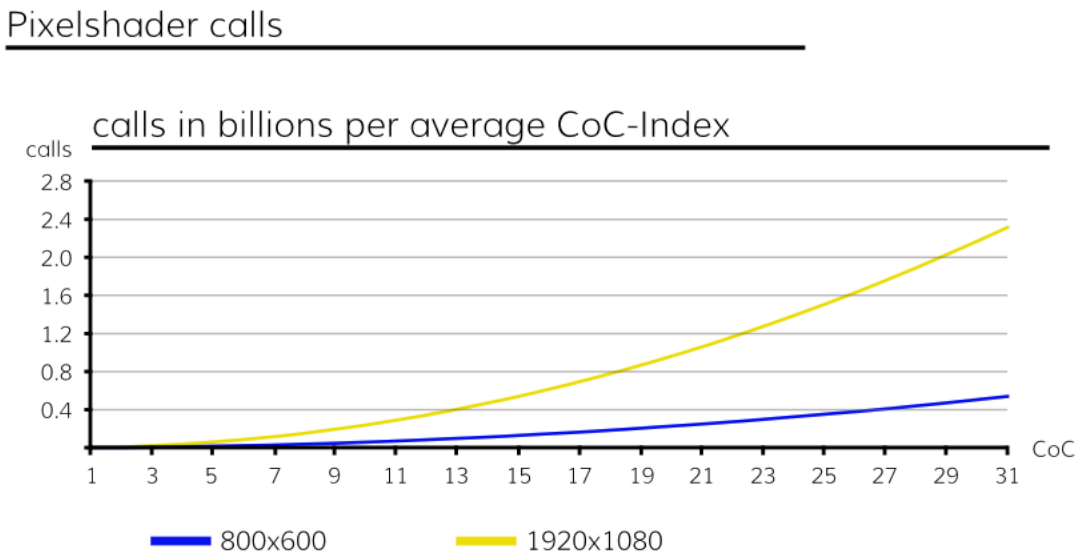

Figure 5.1: A chart showing the impact of the chosen resolution as pixelshader calls in billions by average circle of confusion diameter.

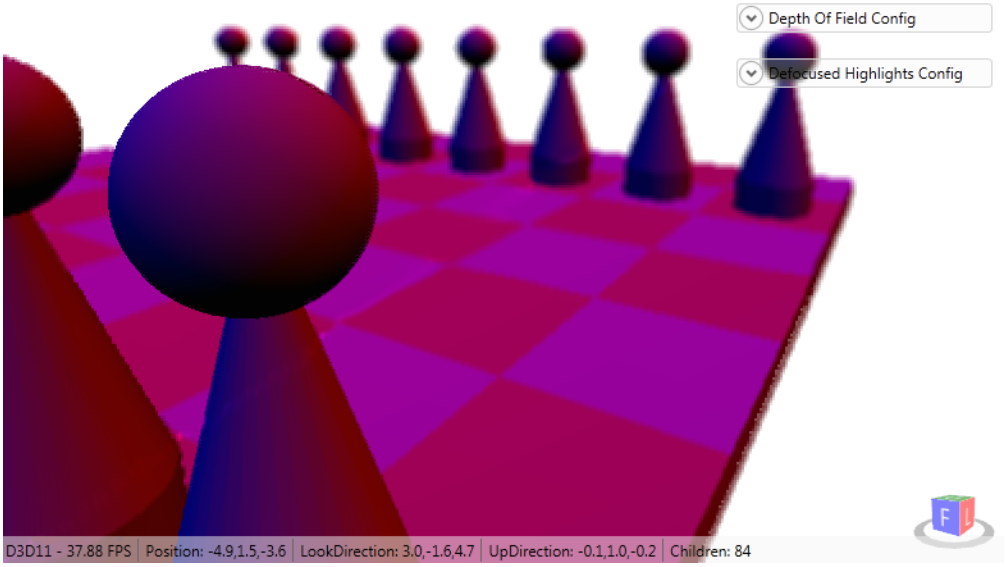

Figure 5.2: Application of depth of field algorithm on a simple chess field.

With incrementing the circle of confusion diameter, the required calls in the pixel shader is increasing quadratically. A higher resolutions leads to strong impact on the real-time functionality, heavier than the actual average diameter for the expansions.

Mobile graphic cards and graphic chips are as performant as the technology of seven years ago. The current high performance components can already handle an average circle of confusion diameter of up to fourty. This values are computed refering to the depth of field effect on precomputed images with the according depth values. Adding shader programs to compute the image is additional load to the framerates described in Section 5.1.

Using cheap pre processing rendering algorithms, good performance can be achieved including this effect. Figure 5.2, 5.3, 5.4, 5.5, 5.6 shows the application of the depth of field rendering algorithm on a chessfield with pawns.

5.1 Framerates

The framerates for this effect are computed by removing all rendering procedures apart from the depth of field algorithm. This is done because it is completely independent of any predecending applications as long as there is color data and the depth information present. Shown in Figure 5.7, the framerates are sampled by rendering a huge amount of pixels that are expanded to sprites as in Section 3.2, simulating a complex scene with averaged circle of confusion diameters.

The local minimum visible in this figure is due to the change from diameter three to four as the quarter reduction from Section 3.6 starts acting. Taking the different scales on the abscissaes into consideration, using the quarter reduction, real-time framerates

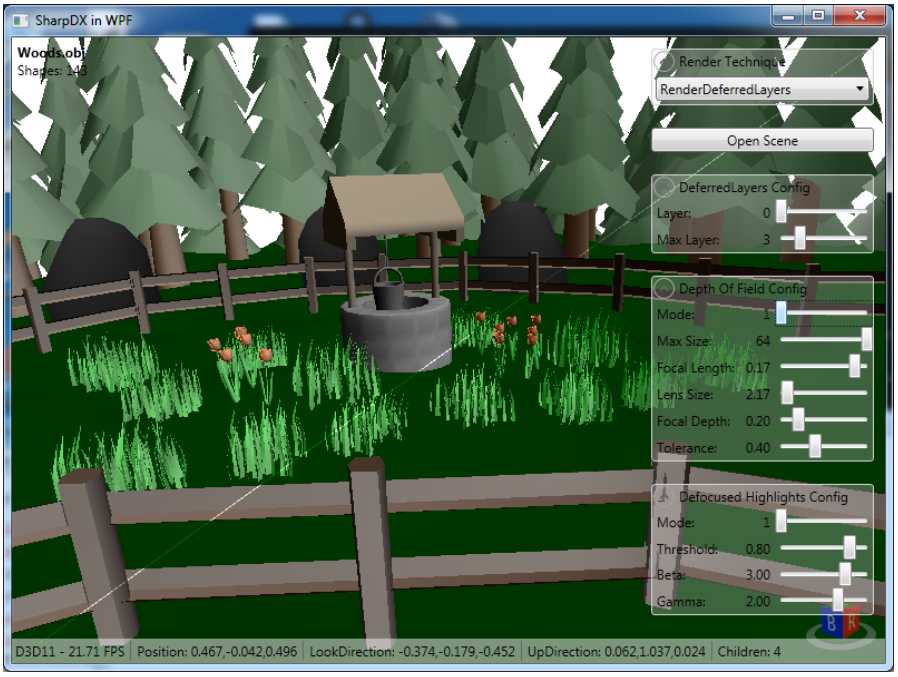

Figure 5.3: Scene in the woods: sharp with no depth of field applied.

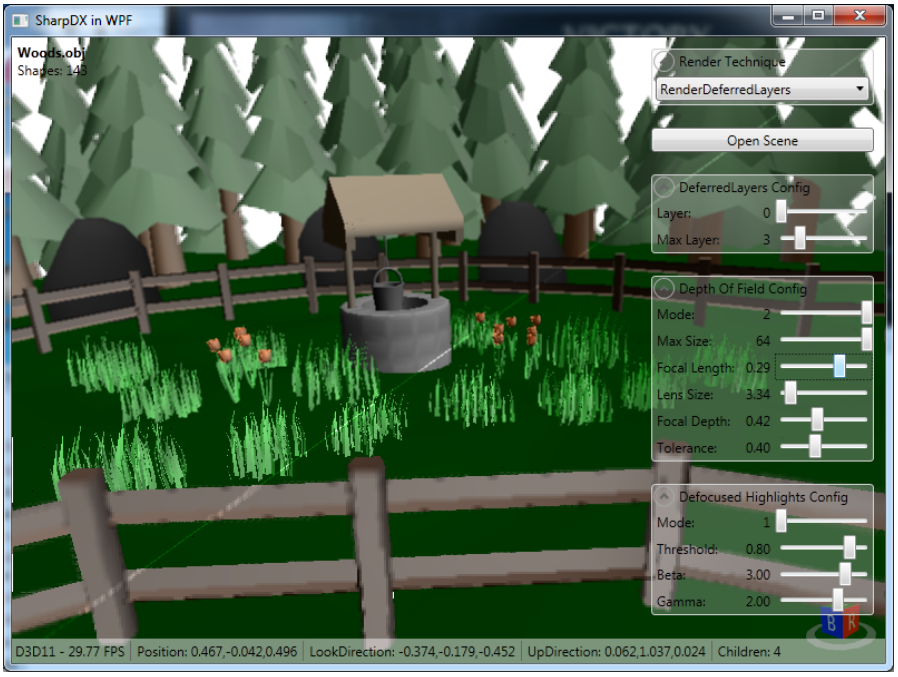

Figure 5.4: Scene in the woods: depth of field enabled, grass is focused.

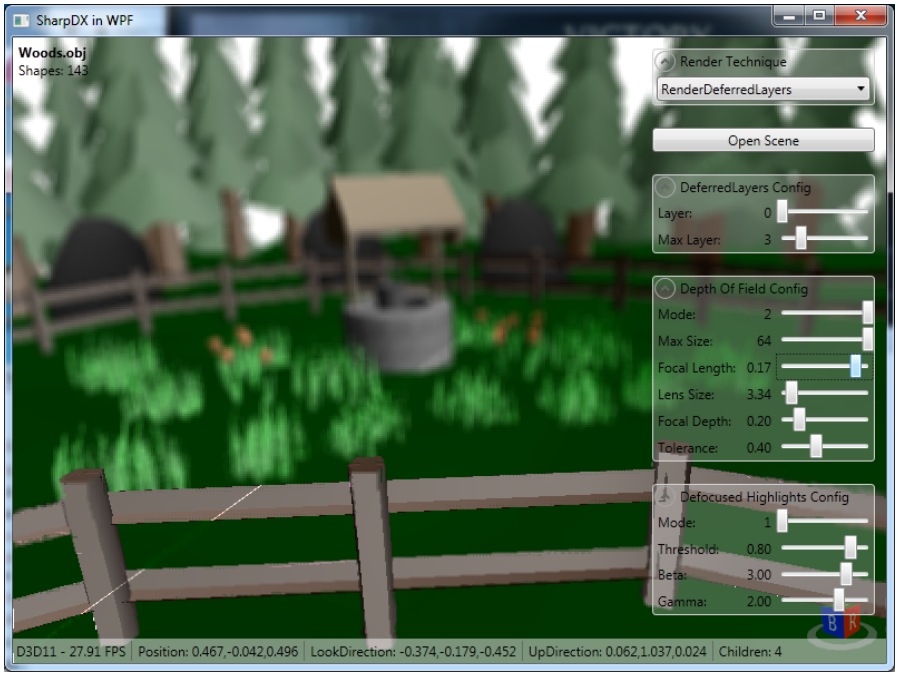

Figure 5.5: Scene in the woods: depth of field enabled, fence is focused.

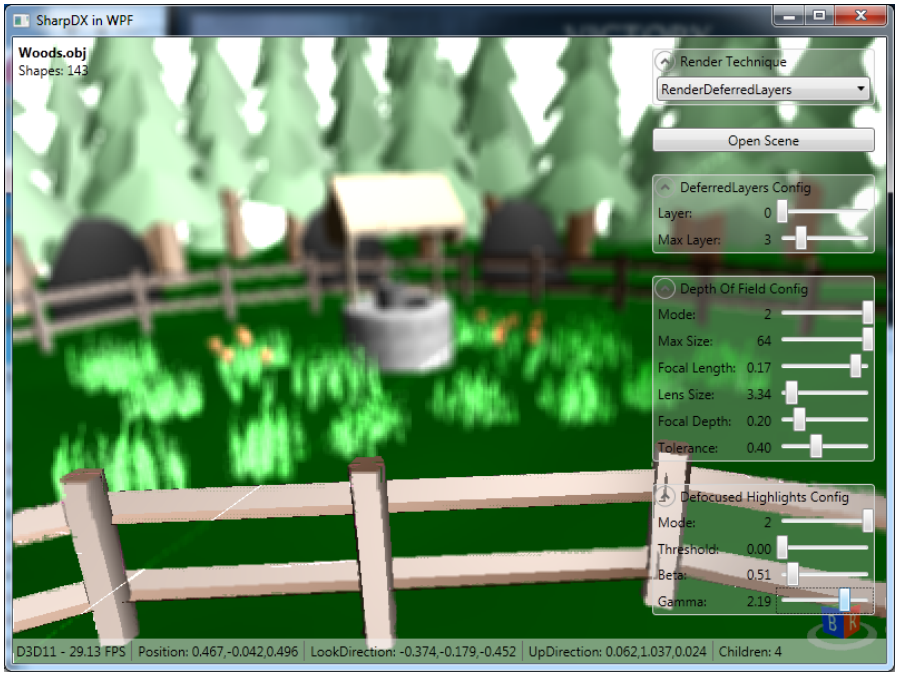

Figure 5.6: Scene in the woods: depth of field enabled, fence is focused and defocused highlights are on.

until a circle of confusion diameter of 38 are achieved for the resolution 800x600. Testing the depth of field rendering algorithm on a full hd screen, the effect remains stable until the index of sixteen.

5.2 Conclusion

The depth of field rendering algorithm as it is implemented in 2008 [LKC08] was primarily interesting because of the intuitive procedure to create the desired effect. To transform all pixels into sprites and expanding them to individual circle of confusion diameter sizes took advantage of the new texturing extension of the point sprites from DirectX9. Doing this expansion manually seemed to be too expensive, but with the hardware optimizations of this internal implementation the idea was realized. Without the quarter resolution optimization, the effect was too slow for real-time applications. Only because of the reduction the framerates could advance from five to thirty on an average circle of confusion diameter of sixteen. This value bases on a screen resolution of 1024x768 and is comparable with the todays performance on a fullHD 1920x1080 with the same expansion size.

It is possible to add any desired intensity distribution function with the contributed bokeh preprocessing. Any image will be distributed into its discrete versions before the rendering of the depth of field algorithm occures. Therefore, there is no additional effort required to replace the intensity distribution function. The additional contribution of the bokeh normalization saved lots of operations at the cost of acceptable memory overhead. Taking a fullHD resolution and an average circle of confusion diameter of sixteen into consideration, about 33 million operations per frame are saved. The reduction of computational cost increases with the resolution, enabling the rendering algorithm for greater pictures.

Nowadays, the performance gain using this acceleration technique from the paper [LKC08] remains the same. Without this step, the effect remains seven years later not practicable as the framerate starts to decrease dramatically starting from a circle of confusion diameter of six. A possibility to increase the effects efficiency would be, to apply this quarter reduction in a greater style. For example, remain the full resolution for circle of confusion diameter 1-3, apply once for 4-7, apply twice for 8-15, apply tripple for 16-31 and so on. The impact could be strong aliasing artifacts due to the magnification resizes. Since the performance expanded greatly over time, the depth of field rendering algorithm idea must wait a few more years to reach acceptable performance for application.

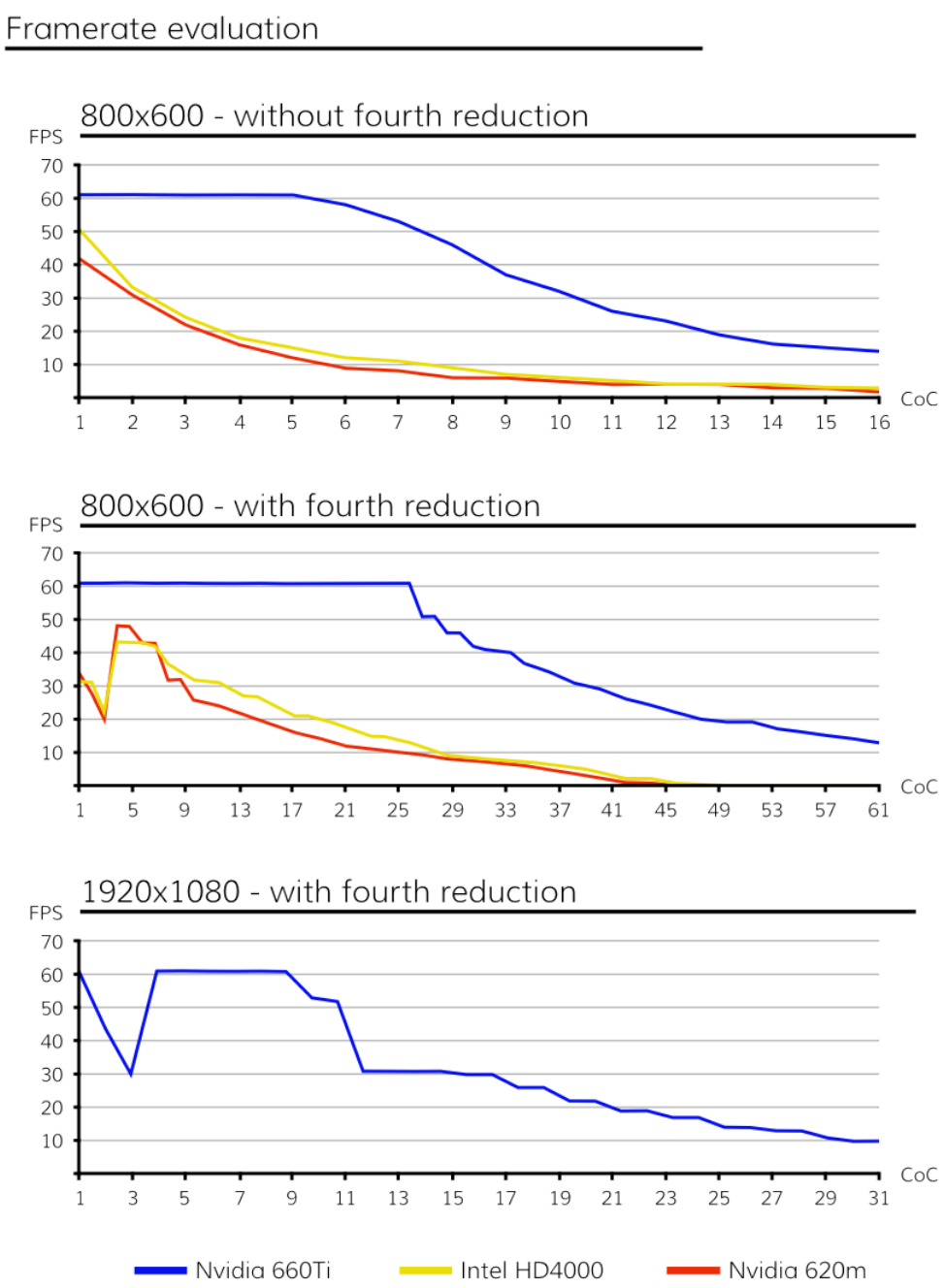

Figure 5.7: The framerates with their respective average circle of confusion (CoC) diameters. The resolution means, that 800x600=480.000 and 1920x1080=2.073.600 pixels are transformed into sprites and expanded to the given sizes on the axis of abscissae. The quarter reduction starts at a diameter of four and is visible in the diagram as local minimum.

Bibliography

[BBGK02] Brian A. Barsky, Adam W. Bargteil, Daniel D. Garcia, and Stanley A. Klein. Introducing vision-realistic rendering. *In Proc. Eurographics Rendering Workshop*, pages 26–28, 2002.

[BW02] Juan Buhler and Dan Wexler. A phenomenological model for bokeh rendering. *ACM SIGGRAPH abstracts and applications*, 2002.

[HA90] Paul Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. *ACM SIGGRAPH Computer Graphics*, 24(4), 1990.

[KTB09] Todd J. Kosloff, Michael W. Tao, and Brian A. Barsky. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. pages 39–46, 2009.

[Lan02] Hayden Landis. Production-ready global illumination. *ACM SIGGRAPH Course Note*, 2002.

[LES10] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Real-time lens blur effects and focus control. *ACM Trans. Graph*, 29(4):65:1–65:7, 2010.

[LKC08] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using point splatting on per-pixel layers. *Pacific Graphics 2008*, 27(7):1955–1962, 2008.

[LKC09] Sungkil Lee, Gerard Jounghyun Kim, and Seungmoon Choi. Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation. *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, 15(3):453–464, 2009.

[MvL00] Jurriaan D. Mulder and Robert van Liera. Fast perception-based depth of field rendering. *In Proc. VRST*, pages 129–133, 2000.

[PC82] Michael Potmesil and Indranil Chakravarty. Synthetic image generation with a lens and aperture camera model. *ACM Transactions on Graphics*, 1(2):85–108, 1982.

[SW12] David Schedl and Michael Wimmer. Simulating partial occlusion in post-processing depth-of-field methods. *Journal of WSCG*, 20:239–246, 6 2012.

[YWY10] Xuan Yu, Rui Wang, and Jingyi Yu. Real-time depth of field rendering via dynamic light field generation and filtering. *Pacific Graphics*, 29(7), 2010.