UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Siim Anderson

Stylized 3D Depth of Field

Bachelor's Thesis (9 ECTS)

Supervisors: Raimond-Hendrik Tunnel, MSc
Santiago Montesdeoca, PhD

Tartu 2021

Stylized 3D Depth of Field

Abstract:

In this thesis a stylized depth of field effect is developed that would have different options for stylization. Four depth of field algorithms are implemented in the process and one of them is chosen to be used in the end result. This thesis is in collaboration with the computer graphics company Artineering and the effects are implemented for use in their 3D graphics engine Flair. Potential users of Flair, depth of field algorithms and the stylization effect tested the implementations and assessed their aesthetic quality as well as usability.

Keywords:

Depth of Field, real-time rendering, blur, computer graphics, Flair, Autodesk Maya, post-processing effect

CERCS:P170 Computer Science, numerical analysis, systems, control

Stiliseeritud 3D teravussügavus

Lühikokkuvõte:

Bakalaureusetöö eesmärgina loodi stiliseeritud teravussügavuse efekt, millel oleks erinevad võimalused stiliseerimiseks. Töö käigus implementeeriti neli teravussügavuse algoritmi, millest üks valiti lõpu stiilis kasutamiseks. Tulem valmis koostöös arvutigraafika firmaga Artineering, mistõttu implementeeriti efektid nende 3D graafika mootoris Flair. Implementatsioone testisid potentsiaalsed Flair'i, teravussügavus- ja stilisatsiooniefektide kasutajad, kes hindasid nende esteetlikku kvaliteeti ja kasutatavust.

Võtmesõnad:

Teravussügavus, reaal-ajas renderdus, hägustus, arvutigraafika, Flair, Autodesk Maya, järeltöötluse efekt

CERCS:P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1 Introduction

Depth of Field or DoF for short, is the effect of blurring out the projections of the objects that are not in a given range of distances, e.g. are too far away or too close from a certain focus point [1]. Humans typically do not pay attention to this with their own eyes, due to the fact that the eye naturally adapts to focus on the object that is being viewed. This effect can be perceived much better when we are looking at photos and videos due to the camera's focus lying elsewhere and the blurred part being visible[1].

Since computers do not use lenses to synthesize an image (*render*[2]) they must use algorithms to achieve the depth of field effect. These algorithms, that emulate the DoF effect on every pixel, produce similar results in a full render as one would get with a physical camera. Popular 3D modelling softwares like Nuke[3], Autodesk Maya[4] and Blender[5] provide visual effects artists the tools needed to produce such an effect. The depth of field can also have a certain style to it. One such software called Flair, that is being developed by Artineering[6], is currently missing an implementation of such an effect. This thesis is about finding the best DoF effect that is stylized to be used in Flair and implementing it there.

Chapter 2 explains what is DoF and all the necessary terms that are needed. Chapter 3 gives an overview of the technologies being used to implement the algorithms. Chapter 4 shows a survey of algorithms that can be used in Flair and also an implementation of them. Chapter 5 uses this algorithm to show a stylized depth of field effect to be used in Flair. Chapter 6 shows the analysis of the usability testing phase. Finally, chapter 7 gives an overview and the work done and concludes this thesis.

The implementation of the algorithms are in Appendix I Repository. The Glossary contains all the necessary term definitions. The questionnaire and manual are presented with the Accompanying Files.

[1] https://catlikecoding.com/unity/tutorials/advanced-rendering/depth-of-field/
[2] https://graphics.fandom.com/wiki/Rendering
[3] https://learn.foundry.com/nuke/content/
[4] https://knowledge.autodesk.com/support/maya/
[5] https://docs.blender.org/manual/en/latest/render/eevee/introduction.html
[6] https://artineering.io/software/Flair/

2 Depth of Field

To understand DoF an explanation of how real world cameras work is useful to understand. Cameras work by first letting light rays come through an opening (*aperture*[7]). These light rays then hit the *lens*[8] of the camera. The rays then refract and focus due to lenses being curved. In a perfect lens all the rays would converge in a spot called the *focal point*[9]. The distance of the point from the lens is the *focal length*[10]. Depending from how far the rays are coming from they might not refract in a singular point but rather as an area known as the *circle of confusion*[11] (CoC).

The CoC is usually a circle but depends on the aperture shape and can have different forms. This visual representation in the shape of the CoC on the image is known as *bokeh*[12]. The size of the CoC is dependent on the focal length and aperture size. When the focal length is shorter the rays bend more sharply and are focused in a shorter distance while the opposite happens with a longer focal length. Thus regulating it is a good way of controlling the focus area range. When the focal length, aperture shape and diameter are adjusted in a combined effort then they can produce various aesthetic results.

The depth of field effect is produced when rays do not hit the focal point and thus are seen as being blurred, e.g. rays coming from far or near objects (See Figure 1 for a depiction of the bokeh and blurred out effect on a photograph).

[7] https://en.wikipedia.org/wiki/Aperture
[8] https://en.wikipedia.org/wiki/Lens
[9] https://en.wikipedia.org/wiki/Focus_(optics)
[10] https://en.wikipedia.org/wiki/Focal_length
[11] https://www.masterclass.com/articles/a-basic-guide-to-circle-of-confusion-in-photography
[12] https://www.nikonusa.com/en/learn-and-explore/a/tips-and-techniques/bokeh-for-beginners.html

Figure 1. *Middle ground is in focus*[13]

These same results can be replicated in computers. A simple way of understanding how DoF works is that for each pixel their CoC is calculated. Most DoF algorithms use some sort of CoC formula. The classic formula is defined as:

$$c = A \cdot \frac{|S_2 - S_1|}{S_2} \cdot \frac{f}{S_1 - f} \tag{1}$$

where S_1 is the focus distance, S_2 is the given pixel depth, A is the aperture size, f is the focal length and C is the size of CoC. *Figure 2* shows a graphical representation of the CoC size.

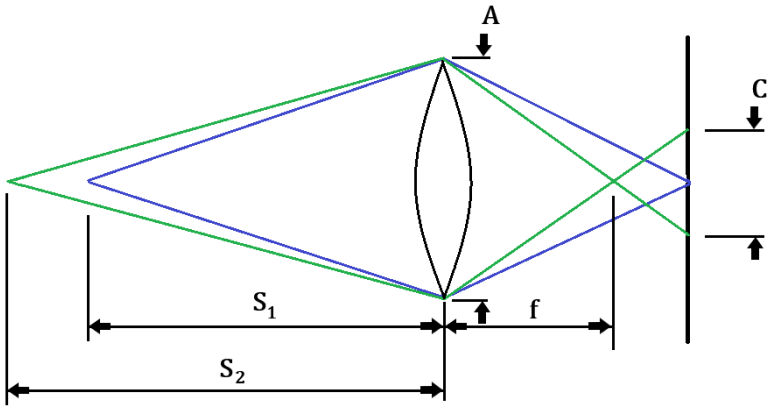

Figure 2. CoC depiction.

The neighbouring pixels that fit inside the radius of the CoC are used to construct a matrix. This image piece is then combined with a pre-calculated blur matrix called *kernel* in an

[13] https://www.nikonusa.com/en/learn-and-explore/a/tips-and-techniques/

operation known as *convolution* [2]. This is not a traditional matrix multiplication however. The blur kernel is first by rotated 180°. After that the values with similar positions from the matrices are multiplied and all the values are summed. This sum is the center pixels final value. In order to blur the image the center pixels value has to be divided by the sum of the kernel values (This is described in chapter 4.1). The equation for convolution is seen Equation (2) where the minus sign flips the kernel by 180° .

$$w(x,y)\circledast f(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)\cdot f(x-s,\ \ y-t) \qquad (2)$$

As hardware has developed new and more complex 3D-modelling softwares have been developed. These graphical softwares use different editing tools to implement multiple effects, including Depth of Field.

3 Used Technologies

The main tool used in this work is Flair. However, Flair shines best when used in conjunction with Autodesk Maya, another 3D rendering software. To understand the implementations in later chapters an overview of Maya and Flair is given to show how their workflow works.

3.1 AutoDesk Maya

Maya is in the forefront of creating convincing and realistic 3D digital content which includes models, animations, visual effects, games and simulations. By 2015 it had been used in every Oscar winning movie since 1997[14]. This shows how widely used and relied on it is in the film industry.

In Maya the users work in a virtual workspace called *scene.* All of the elements in a scene are node-based and have their own attributes and customization options. This creates a hierarchy in which nodes are dependent on each other and provide information to other nodes[15]. The element hierarchy can also be understood as a dependency graph where nodes are connected via edges[16].

Another feature of Maya is the possibility to install extensions to the software as well. These plugins are used for various purposes, such as modelling, animation and rendering. Some of these plugins interact with external applications, i.e. Flair.

3.2 Flair

Flair is a node-based real-time graphics engine developed by Artineering. Typical 3D modelling software requires everything to be imported into the engine and then the image-processing has to be hard coded in the form of *shader. This* is difficult for people who do not know how to do this, e.g. non-technical artists.

In animation and visual effects (VFX[17]) the scene is rendered first and after that the stylization can be added to the render. This is where Flair comes in. In Flair it is possible to modify an object's vertices and re-render them in real time with an artist-friendly toolset.

[14] https://venturebeat.com/2015/01/15/hollywood-fx-pros-i-want-to-be-an-oscars-maya-winner/
[15] https://en.wikipedia.org/wiki/Autodesk_Maya
[16] https://en.wikipedia.org/wiki/Dependency_graph
[17] https://www.studiobinder.com/blog/what-is-vfx/

This graphics engine can be used as a standalone or as an extension for applications like Autodesk Maya.

Controlling the image-processing and the act of stylizing a scene or image is achieved through a node graph (See *Figure 3*). This node-based workflow gives artists the freedom to make their own stylizations in real-time and they are able to add information into the *object-space* itself to see results right away.

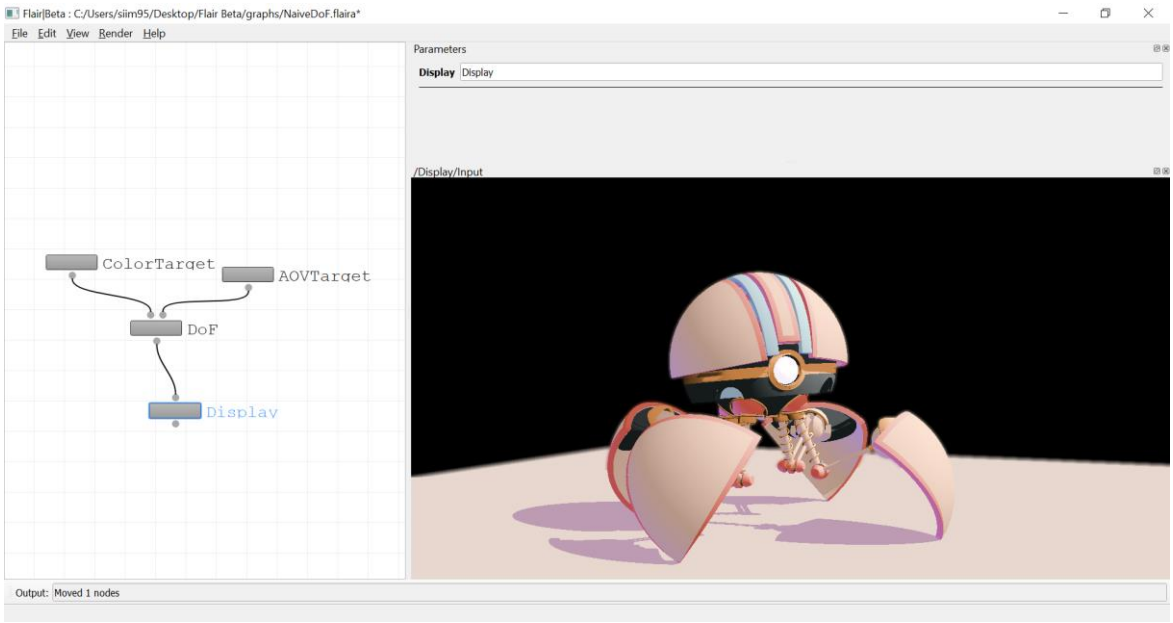

Figure 3. Flair's node graph.

The workflow is simple. To get images either `Read` or `Import` nodes are used. Read nodes use images that are saved on the hard-drive (HDD) while Import nodes use images that other applications (i.e. Maya) have saved on the GPU shared memory. These nodes are then connected to `Shader` nodes that use the pre-coded shader files in the GLSL shading language. It is also possible to modify the GLSL code within the application itself. When the image-processing has been done then these images can either be saved on the hard-drive or sent back to another application. Thus these connected `Shader` nodes provide the stylization for a given scene or image. With the help of algorithms there can be multiple effects that a scene can have, i.e. Depth of Field.

4 Depth of Field Algorithms

In this chapter depth of field algorithms are shown. All of the algorithms have been chosen to be usable in real-time rendering. Other possible candidates were considered for this work as well. For instance, Practical Gather-based Bokeh Depth of Field by Wojciech Sterna [3] or David C. Schedl et. al. Simulating Partial Occlusion in Post-Processing Depth-of-Field Methods [4]. However, *downsampling* is not possible in Flair thus Sterna's work was left out. Schedl's work was not included as it was specified in their paper that their implementation was not usable in real-time with their blurring function.

Chapter 4.1 describes a naive approach of blurring out objects that are not in the focus area. All of the following chapters show algorithms that are from papers published in scientific magazines and journals. First a general description of an algorithm is given and then the implementation details and results in Flair are presented.

4.1 The Naive Approach

This approach is implemented to give a better understanding of how blurring is done in combination with depth information.

A depth threshold parameter is defined in the shader code that can be modified in the Flair interface as a slider. If the depth of a given pixel is larger than the defined depth threshold parameter then a simple *box blur* is applied, otherwise it is in focus.

Box blur works by performing a convolution (See Equation (2)) operation with the image and blur kernel [2]. The resulting value is divided by the sum of all the blur kernels values (*weights*). The box blur kernel has equal weights for all of the pixels and thus the result is divided by 9 (See *Figure 4*).

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Figure 4. Box blur kernel with equal weights.

The radius of the kernel is defined in the code as a modifiable parameter that can change the kernel size (See *Figure 5*).

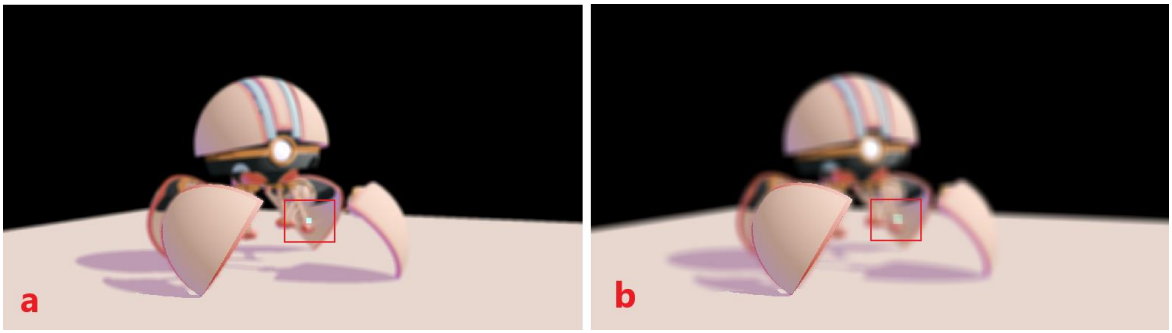

Figure 5. Box blur with a) Kernel radius of 5 and b) Kernel radius of 10.

One major drawback in this approach is that it is non-separable. The horizontal and vertical axis's are blurred at the same time. Looping through both the x and y axis is costly. The time complexity for this is $O(N * r^2)$, where N is the number elements in the kernel and r is the radius of the kernel. If the horizontal and vertical axis's are done separably then the time complexity would lower to $O(N * r)$, improving performance significantly. However, this approach does produce other issues.

This simple implementation produces bugs (*artefacts*). In the images on *Figure 5* an artefact is seen as a blue box (highlighted with a red box). This is a side product of taking every pixel into account that fits inside the kernel. When the kernel is near an edge there is a sudden shift in the pixel values. If that edge overlaps with the background, then these values produce an artefact known as *pixel bleeding*. The colour of one pixel is affecting the colour of the background pixel. This is also known as intensity leakage (See Figure 6).

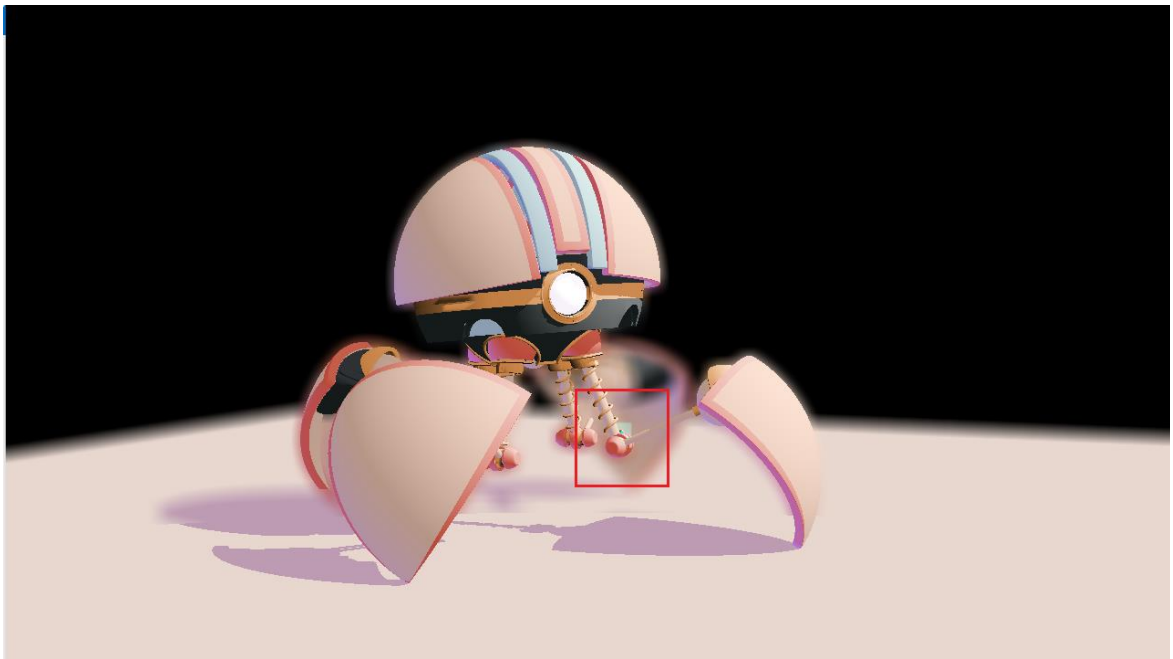

Figure 6. *Highlighted in the image the blue pixel is bleeding into the background.*

A similar approach to the box blur is a Gaussian filter[18]. The difference between them is that a Gaussian filter uses unequal weights that are calculated with a Gaussian function. Although a Gaussian filter produces smoother results, it also has pixel bleeding the same as a box blur. The Gaussian filter can be implemented separably to improve performance.

4.2 Accurate Depth-of-Field Rendering Using Adaptive Bilateral Depth Filtering

S. Wu et. al. present a novel DoF algorithm [5]. They use an adaptive bilateral filter which replaces the widely used Gaussian filter. Due to the fact that their method uses bilateral filtering then they have an edge preserving feature (See *Figure 7*).

First, a pre-rendered depth map and colour image are needed. The depth map holds depth information for all of the pixels. It is needed for the CoC and final blur calculations.

The CoC values of each pixel are calculated and stored as a *texture*. That texture is passed to the bilateral filter, which then calculates the blur amount.

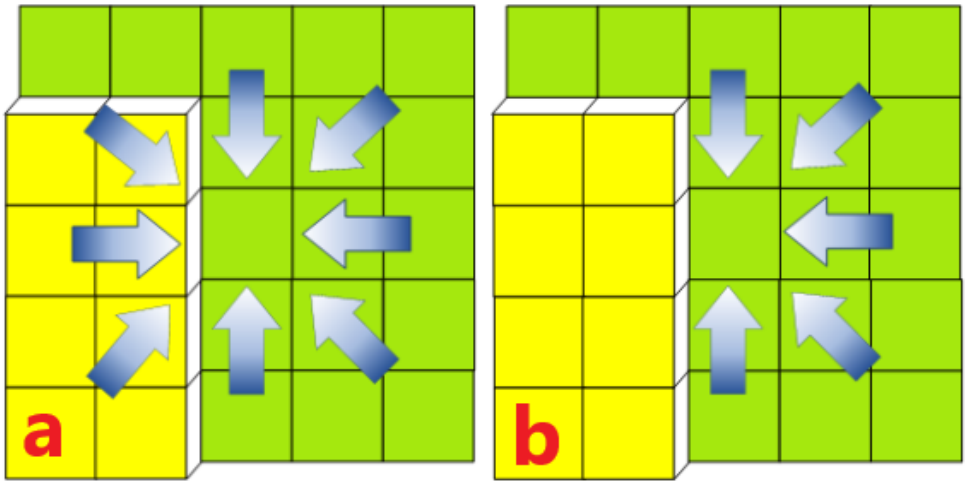

Figure 7. a) Gaussian filter and b) Bilateral filter gathering methods near an edge.

This implementation of the Adaptive Bilateral Depth Filter uses the bilateral filter alone. Due to the fact that it is not specified in the paper what function should be used for the focus area, the adaptive part of the filter was not utilized in this thesis.

The depth map and colour image are pre-rendered in Maya. They are then imported in Flair with the import node.

[18] https://en.wikipedia.org/wiki/Gaussian_filter

First, the CoC map is calculated in a single node with three parameters that can be modified in the Flair interface: `FocalLength`, `LensDiameter` and `FocusRegionDepth`. The map is passed down to the `AdaptiveBilateralFilter` node for the final result (See *Figure 8*).

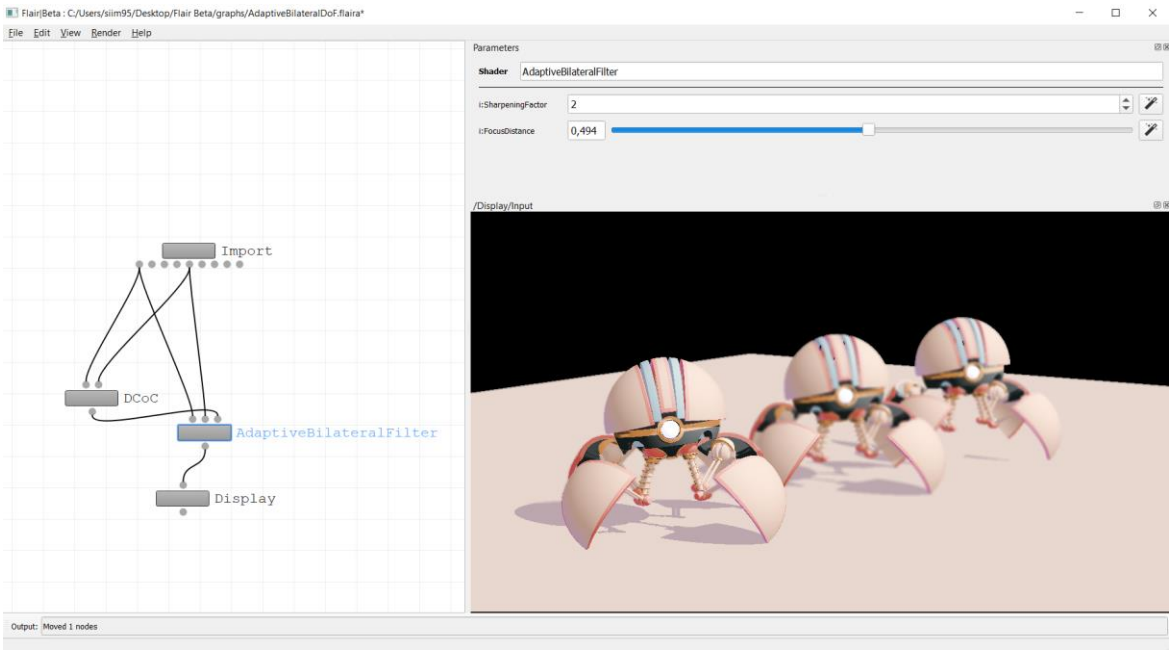

Figure 8. Final result of the Bilateral filter with the accompanying graph.

The filter node has two parameters: `SharpeningFactor` and `FocusDistance`. These are used to sharpen and regulate the blur distance respectively. This approach does not produce any bokeh effects as it is a simple blurring operation. The bokeh effects are however produced in other algorithms.

4.3 Efficiently Simulating the Bokeh of Polygonal Apertures in a Post-Process Depth of Field Shader

L. McIntosh et. al. describe a novel approach to real-time depth of field rendering [6]. Their algorithm takes advantage of separable filtering to simulate bokeh shapes and accounts for intensity leakage. The HLSL code for this algorithm was included with the research. Since Flair uses shaders in GLSL then the code had to be translated into the correct coding language.

The first step in their approach is to calculate the CoC values. The CoC value is stored in the alpha channel of the Colour map texture. As the CoC value is stored in the alpha channel, then the output is the same as image inputted. The final result is clamped between zero and a maximum CoC diameter to prevent artefacts.

Next *sampling* offsets are created. These sampling offsets can be understood as a blur kernel that is laid on the image. A uniform kernel is used so that the offsets are evenly weighted. To create different bokeh shapes the offset kernel has to be rotated and applied in two separate passes. This creates a cumulative effect and allows to create more complex parallelograms. It is specified in the paper that the values should be calculated in a separate shader file. However, as the offsets are stored as an array, it is not possible to do this in Flair. For this reason, the kernel values are calculated in the shader file for the filtering passes.

In this implementation the filtering pass starts by calculating the kernel offsets. The angle of the kernel is specified in the Flair interface from a slider After that the algorithm calculates the amount of blur for each pixel based on the offset samples.

To avoid intensity leakage, the depth of the offset sample is compared with the current pixel depth and the same is done with the CoC. If both of them are less than the current pixel depth and CoC values, then this sample is ignored as it would contribute to pixel bleeding.

The values that are not ignored are added to a running total. When all of the offsets have been sampled then the running total is divided by the number of valid samples found. This average is then returned for use in the final pass.

The last stage is used to combine two images. This can be thought of as a boolean operation, e.g. union or intersection. Two functions that perform similar results in GLSL are `Min(x, y)` and `Max(x, y)` respectively.

For the filtering passes this implementation uses the first and second pass once. This resulted in a smaller blur and an increase in brightness (See *Figure 9*).

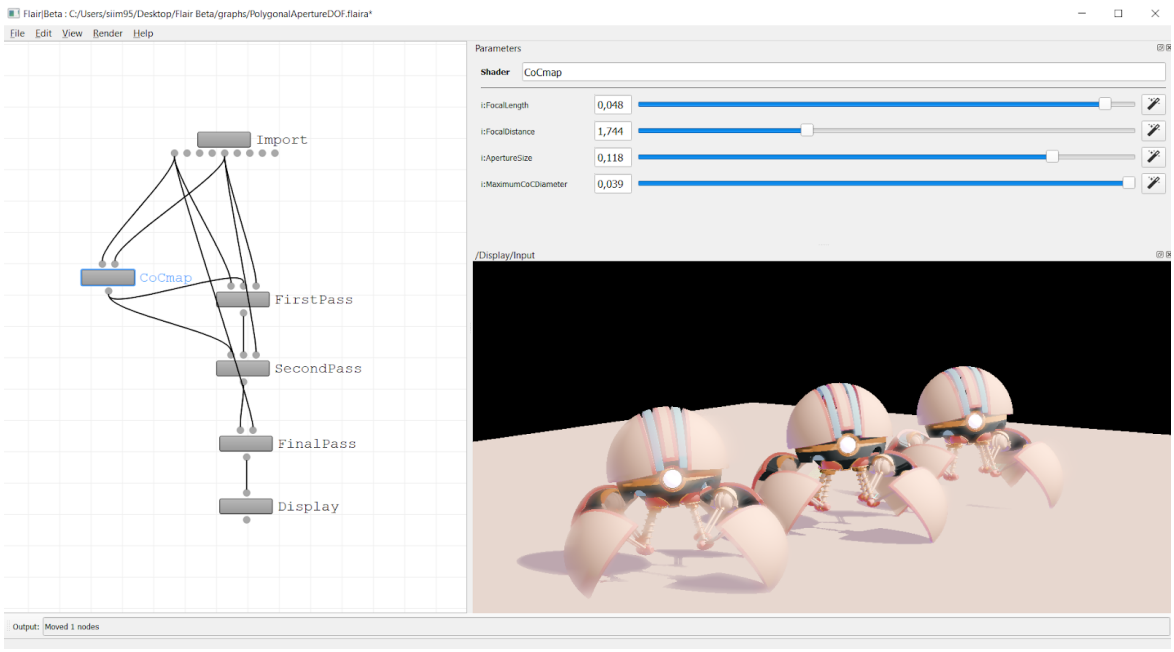

Figure 9. The final result of the filtering passes with the middle bot in focus.

It was pointed out in the paper that `Min(x,y)` or `Max(x,y)` can be used in the final pass. This implementation uses the `Max` function as it was specified that the `Min` function lowers the image intensity. The `Max` function is also the reason why there is an increase in brightness.

4.4 Depth of Field Rendering via Adaptive Recursive Filtering

S. Xu et. al., present a new post-processing method for the depth of field effect [7]. They achieve this by adaptively smoothing the image with their corresponding depth and circle of confusion data. Their filter uses a weighting function used between two neighbouring pixels producing smoothed results. The method handles the borders between the objects that are in-focus and out-focus and also deals with intensity leakage and blurring discontinuity.

The algorithm starts by computing the CoC values as a texture. After the CoC calculations, the scene is segmented into three regions: foreground, focus and background. Two depth thresholds are set D_1, and D_2 such that $(D_1 < D_f < D_2)$. Pixels that are within range $[D_1, D_2]$ are considered sharp and in the focus region. Values that are below D_1 are in the foreground and values beyond D_2 are in the background region. These regions are needed in the next part of the process, weight computation.

Weights are computed based on the region where the current and the neighbouring pixel fall. There are 4 cases that handle the situation as follows:

1. Both of the neighbouring pixels are in the same region (*Figure 10*a).
2. One of the pixels is in the foreground and the other in the focus region (*Figure 10*b).
3. One pixel is in the focus region and the other in the background region (*Figure 10*c).
4. One pixel is in the foreground and the neighbouring pixel is in the background region (*Figure 10*d).

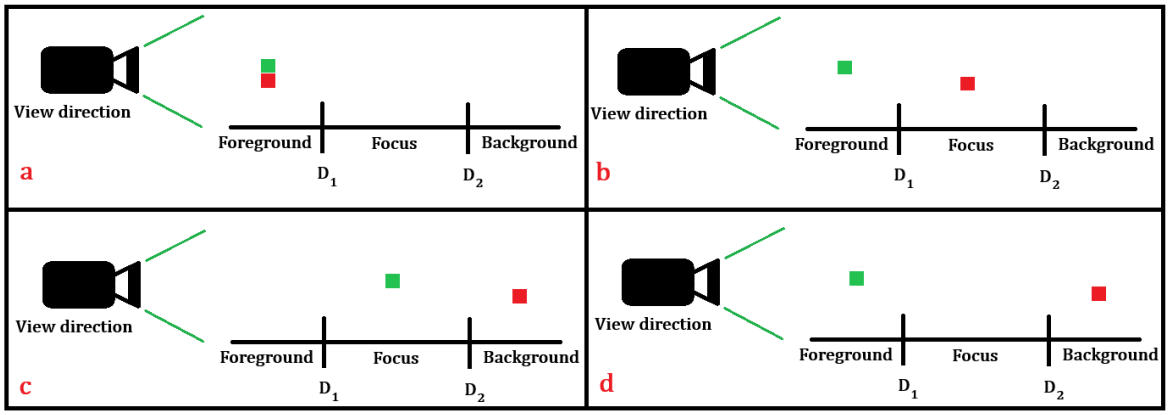

Figure 10. WeightMap cases. Green box is the given pixel and red is the neighbour.

When the weights have been calculated the last step is to use them in the recursive filter. In the process the image is scanned from left to right on the first pass. The opposite is done on the second pass to achieve symmetric results. The best results are achieved in 2-3 iterations.

The implementation of the Adaptive Recursive Depth of Field algorithm starts with rendering the CoC map. It has two parameters, `scale` and `FocalLength`. Regulating these two allows to control the blur amount and focus area distance.

In the research it was specified that the `RegionMap` is rendered in a separate texture. However as there are only two values that are needed then the region distances are calculated in the `WeightMap` shader code.

The `WeightMap` has two parameters, `Cmin` and `FocalLength`. `Cmin` is used to regulate the size of the focus area and `FocalLength` specifies how far the focal plane is (i.e. focus area).

The filtering passes sample the left and right neighbours right after each other. This is done to not offset the image. Thus the first recursive pass samples the left neighbour and the next node the right neighbour. The process is repeated 2 more times to simulate 3 iterations (See *Figure 11* for the final result).

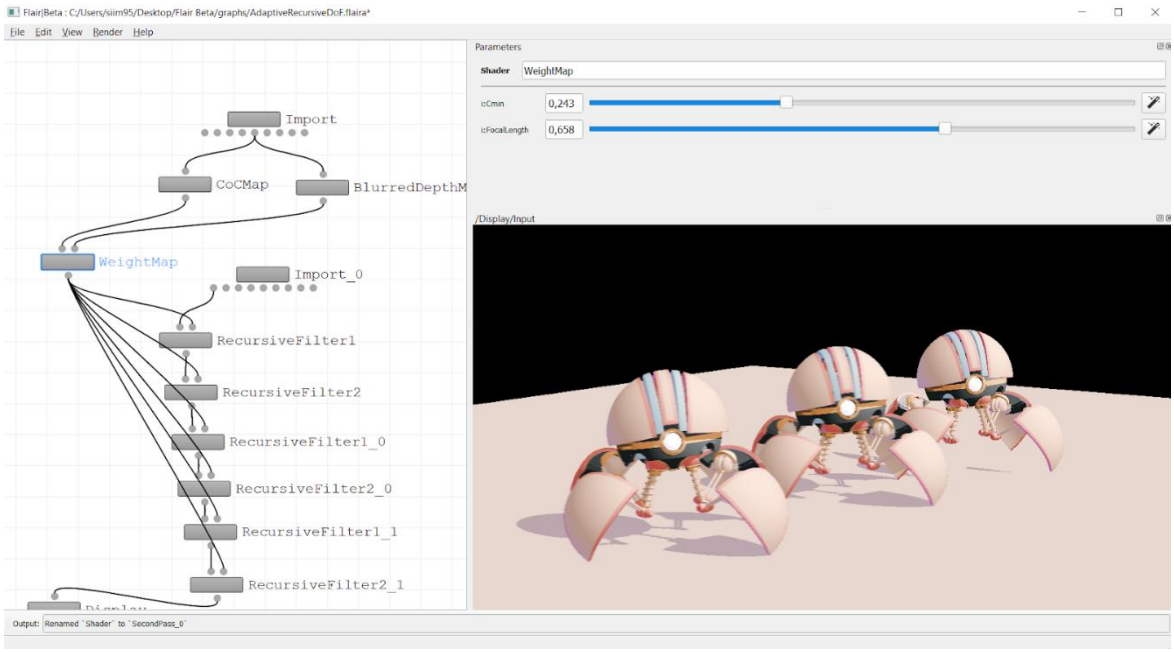

Figure 11. Two bots in the foreground are slightly blurred and the last one in focus.

The `FocalLength` of `CoCMap` and `WeightMap` are the same. They cannot be controlled as a single parameter in Flair thus they are used in both nodes. Thus controlling the focus distance proved difficult.

4.5 Circular Separable Convolution Depth of Field

In his work “Circular Separable Convolution Depth of Field" [8] Kleber Garcia presents a mathematical adaptation and implementation of a separable circular convolution filter. Since the convolution operation is computationally expensive in two dimensions then separating it into two one-dimensional passes greatly increases its speed. Complex numbers are used in calculating the final results on to the screen. Important factors are also low memory usage and large radii circles. This method is used in games like *Madden NFL 17* and *Fifa 17*.

First, the image is segmented into three regions, near, far and the focus area. The near section uses a lower quality filter while the far segment is a high quality filter for blurring. The bokeh effects that these filters produce are seen in *Figure 12*.

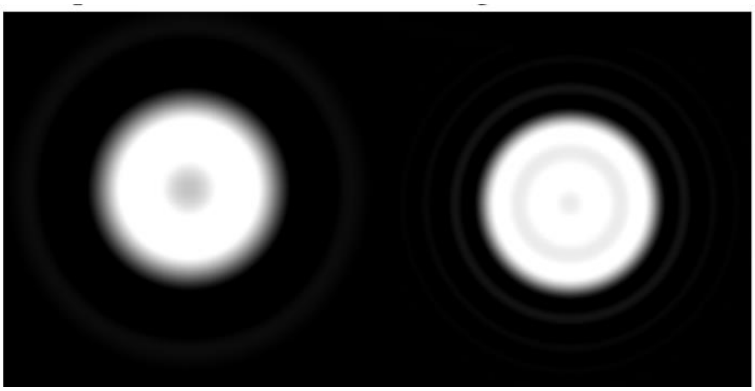

Figure 12. Left is the low quality filter and right is the high quality filter result.

The reason why the near blur is of low quality is because it appears less frequently. The segmented scenes are composited together as the final result.

The code used in this work was implemented by Bart Wronski[19] in his shadertoy implementation[20]. The code blurs the whole image, thus the scene is not segmented into three sections as specified in the paper.

For this work the scene is segmented into two regions, the background and focus area only. This is due to the fact that the blending functions are not known and a smooth transition is not achieved in the scene segmentation (See *Figure 13*).

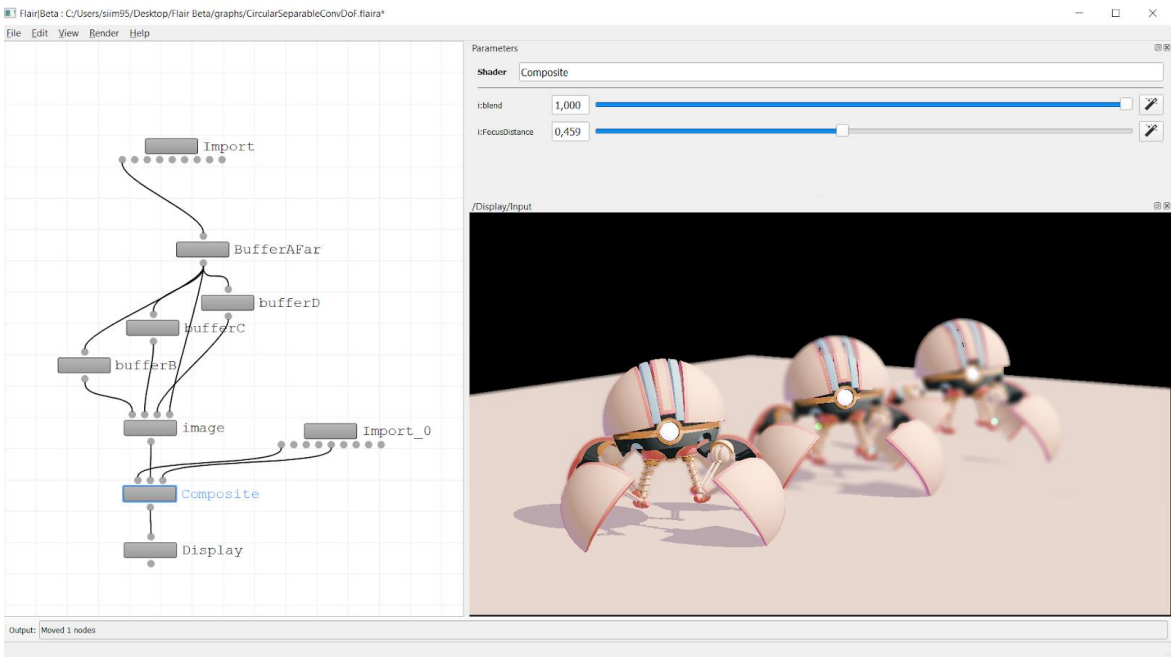

Figure 13. The Circular DoF sharp edge on the middle bot.

[19] https://bartwronski.com/
[20] https://www.shadertoy.com/view/lsBBWy

The code uses the high quality filter for the background blur. The `FocusDistance` can be regulated in the `Composite` node where the focused image and blurred image are blended together. This is done with the `mix` function of GLSL. The alpha value of the blend function can be regulated as a parameter to control the amount of blending. The final result is with a sharp edge blur.

Although there is a sharp edge between the focus and background areas, this algorithm does produce the best blur out of the four. For this reason, this DoF effect is used in further implementations.

5 Stylized Extension

The main goal of this thesis was to develop a stylized DoF effect for use in Flair. This chapter covers what was made and how it was achieved.

Stylization means that different effects are used in combination with depth of field. Inspiration is drawn from *Spiderman: Into the Spiderverse*. There the depth of field effect uses a mix of changing the scene colour and offsetting the characters (See *Figure 14*). This partly looks like a ghosting effect as well. The colours are kept dull as to keep attention on the focus area while also giving a sense of using 3D glasses.

Figure 14. The DoF effect in Spiderman: Into the SpiderVerse.[21]

Thus for the stylized edition of this work a combination of warping and changing colours is used. The warping bends the image to give the viewer a distorted feeling of abstract art. The colour changing is done with a toon shader[22] to make it look more cartoonish. This is also accompanied with the toon shadings effect of having layers of shadetones on the characters (See *Figure 15*).

Achieving toon shading is done with simple calculations[23]. First the *normals* of the texture are needed and a light direction has to be defined. The normals of a surface determine the orientation toward a light source. Since Flair is used in combination with Maya then the *normals texture*[24] can be imported from there. The light direction is defined in the shader code as a `vec3` with x, y and z coordinate as the position in the object space. This light

[21] https://the-avocado.org/2018/06/06/spider-man-into-the-spiderverse-animation-breakdown/
[22] https://en.wikipedia.org/wiki/Cel_shading
[23] https://stackoverflow.com/questions/5795829/using-opengl-toon-shader-in-glsl
[24] https://en.wikipedia.org/wiki/Normal_mapping

position is however fixed to the camera position. When the camera is moved the light direction follows with it. This means that the coordinates are in the camera space.

Using these two vectors a simple dot product can be done to calculate the intensity of the vertex. After that number of shadetones can be defined with if-else statements to check for the intensity. This if-else check returns how intense a pixel colour has to be as such:

```
if (intensity > 0.95)
   colour = vec4(1.0,1.0,1.0,1.0);
else if (intensity > 0.5)
   colour = vec4(0.6,0.6,0.6,1.0);
else if (intensity > 0.25)
   colour = vec4(0.4,0.4,0.4,1.0);
else
   colour = vec4(0.2,0.2,0.2,1.0);
```

The width of the discrete shadow tones can be tuned with adding a multiplier to the intensity.

```
if (intensity > pow(0.95, fraction))
```

`Fraction` is a modifiable parameter in the Flair interface. By raising the intensity to the power of a given number then the shadow width can be modified.

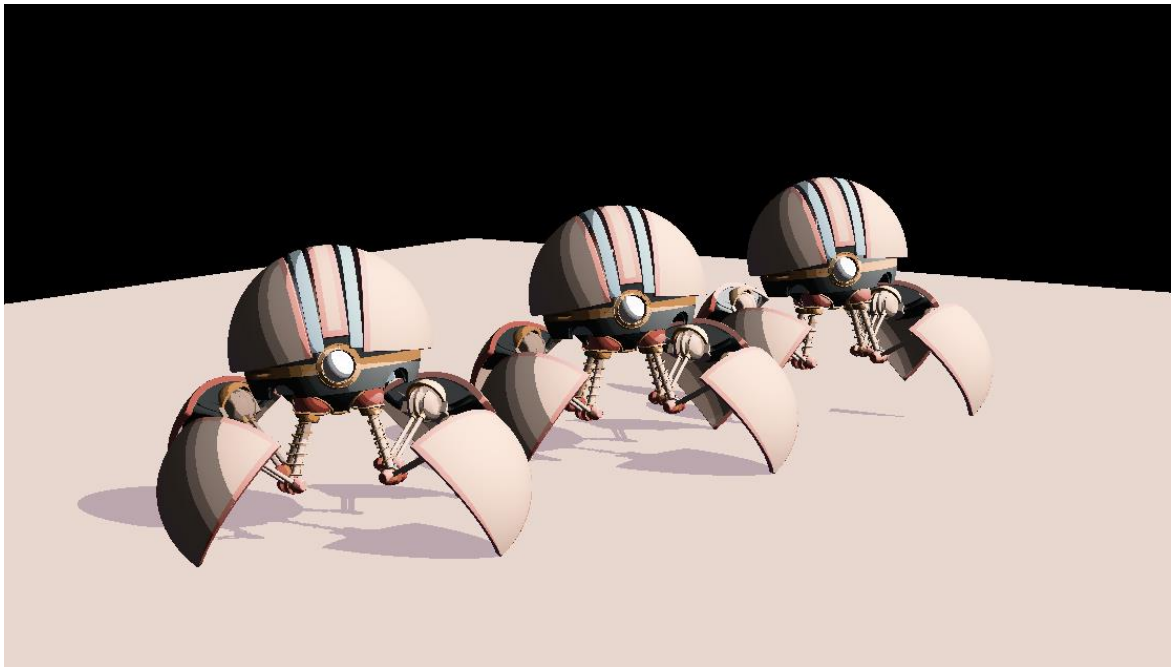

Figure 15. Toon shaded scene with 3 layers of shadows.

The warping effect is from the geeks3D shader library[25]. Warping is seen in the scene as a donut shape where the distance of the rings can be regulated (See *Figure 16*). The actual texture distortion amount can also be modified. The center from which the warp is calculated

[25] https://www.geeks3d.com/20091116/shader-library-2d-shockwave-post-processing-filter-glsl/

is defined as a `vec2` with x and y coordinates. The donut shape is regulated with a `vec3` called `shockParams`. The x and y coordinates of `shockParams` are used for the distortion. The x coordinate is for a minimal effect while the y coordinate is used as a multiplier for the warp. The z coordinate is used to regulate the size of the donut ring. Using this in combination with the `distanceMul` parameter is a great way to control the size of the final distortion.

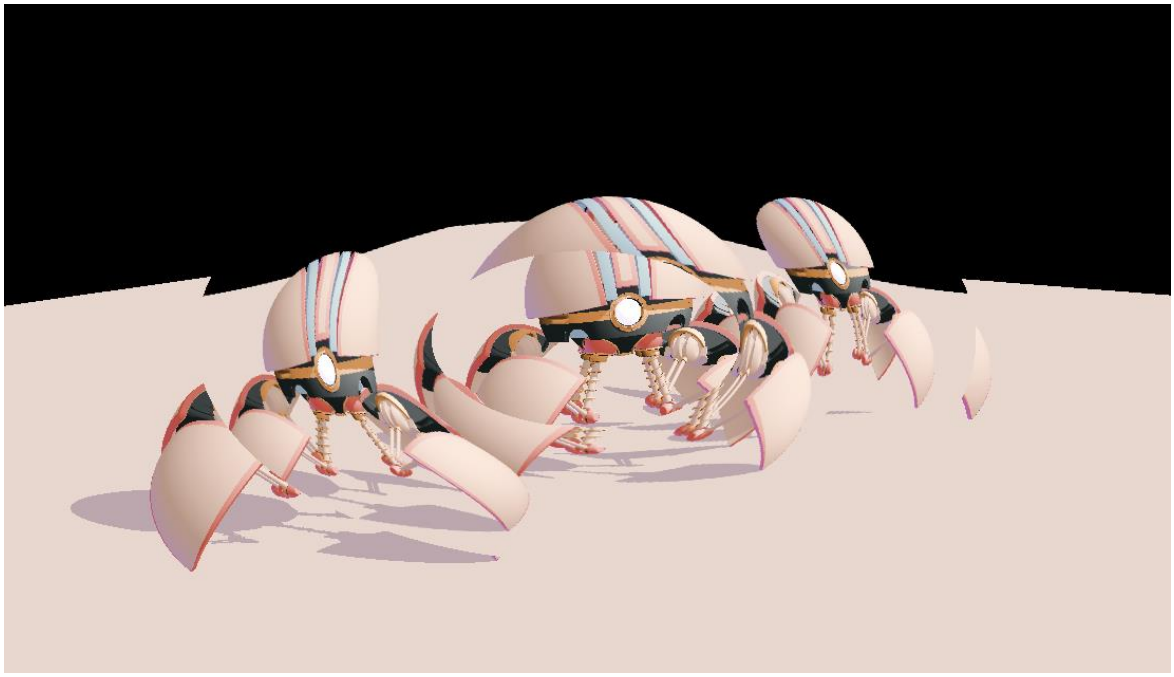

Figure 16. The warping effects donut shape.

Since it is a DoF effect then a blur is used as well. The blur is achieved with the Circular Separable Convolution DoF that is covered in chapter 4.5. All of the previous effects and the blur combined together provide a great degree of artistic variation to the final result seen in *Figure 17*.

Figure 17. Blur combined with warp and toon shading.

This stylization can have future improvements. The colours could be defined in the Flair interface. This would give an artist more control of the toon shader. Warping can be made into another shape: square, circle. This does not have to be in a shape but can be over the entire blur area as well. To find out if potential users would like the current state of the stylization then this has to be tested.

6 Testing

For the algorithms and stylized extension to have any real world users a usability testing phase was needed. Since the implementations and Flair are intended for artists and people who deal with computer graphics then they would be the target audience. This meant finding testers with the right qualifications. Out of the five people who came for the final testing, four were asked straight (through Discord, e-mail) and one person was reached thanks to an e-mail list that was sent out. The questionnaire and raw data are in the Accompanying Files.

First background info about the testers was collected. As the testing involved using people who might use Flair in the future as well, then determining their experience with computer graphics tools and work proficiency was necessary.

The first task was for the testers to reproduce the result from an example image with the intent of finding out whether they could replicate it and if the parameters were understandable. The second task involved using the stylized extension and thus a degree of freedom was given to allow the tester to produce a result of their own liking. The following chapters display the results and analysis of the testing phase.

6.1 Background information

For the background information section, three questions were asked.

First the testers were asked to rate their experience in computer graphics on a scale of 1 to 5, with 1 meaning “No skills” and 5 being an “Expert”.

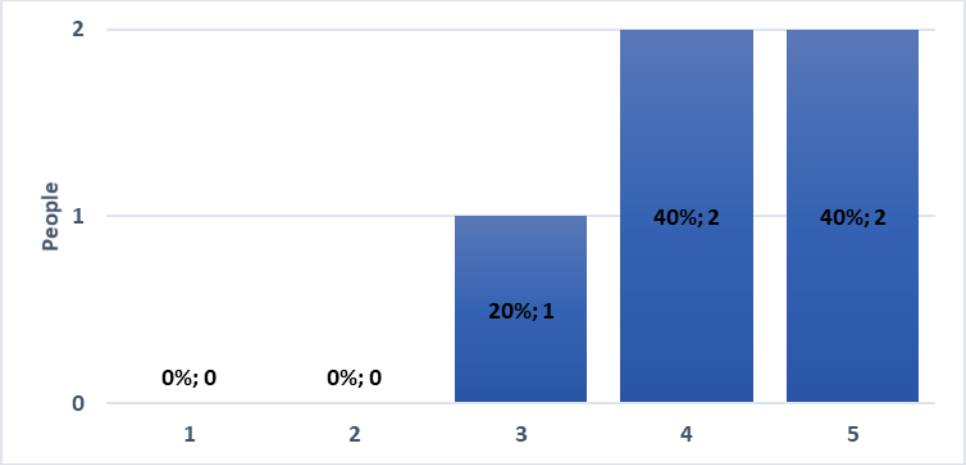

Figure 18. Testers experience in computer graphics.

Due to the fact that the testers work or have worked in the computer graphics industry (gaming development, VFX), then their experience is quite high. This can be seen in *Figure*

18 where they rate their experience highly. This means that their insight into the implementation of DoF algorithms is very valuable.

As Flair is used in combination with Maya, then it had to be determined whether they had any experience with Maya. The testers were asked if they had used Autodesk Maya and rate on a scale of 1 to 5, with 1 meaning "Never used" and 5 "Use it everyday".

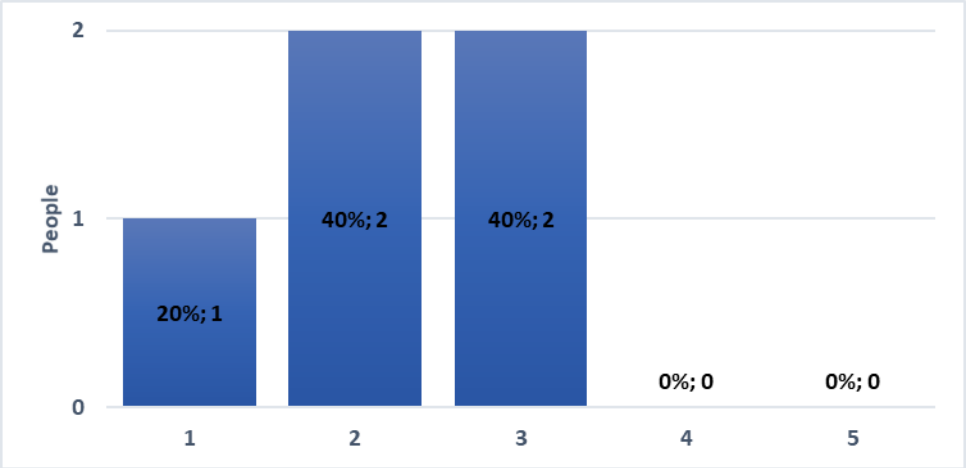

Figure 19. Testers experience with Autodesk Maya.

Figure 19 shows the experience level with Maya can be rated as being below average due to the fact that the testers do not use it very often. A third question was asked to define other computer graphics tools that they were familiar with. The given answers had a lot of variation. The most common modelling software was Blender but similar 3D graphical programs were used as well like Nuke, Houdini and Redshift. Other given answers include Unity, Unreal Engine 4, Godot and CryEngine that are all game engines. Even though the testers were not much familiar with Maya then they are familiar with a lot of the other rendering softwares that are on the market. Thus working with Maya would not be that difficult in this scenario as they would only need to move the camera there.

6.2 DoF Algorithms Usability Testing

Each of the algorithms had the same set of questions.

1. On a scale of 1 to 5 rate how difficult was it to reproduce the same result? (1 – "Very Easy", 2 – "Easy", 3 – "Normal", 4 – "Hard", 5 – "Did not achieve the result")
2. On a scale of 1 to 5 rate the complexity of the parameters? (1 – "Very Easy", 2 – "Easy", 3 – "Normal", 4 – "Hard", 5 – "Did not understand them at all")

3. Were there any artefacts? If yes, please describe them.
4. If there are any remarks about the algorithm, please write here.

The first two questions are linear scales and questions 3, 4 are free to write text paragraphs.

Adaptive Bilateral Filter

The example image given for the Adaptive Bilateral filter algorithm is seen on *Figure 20*. The testers had to figure out how to blur the image so that the background bot and half of the middle bot would be blurred.

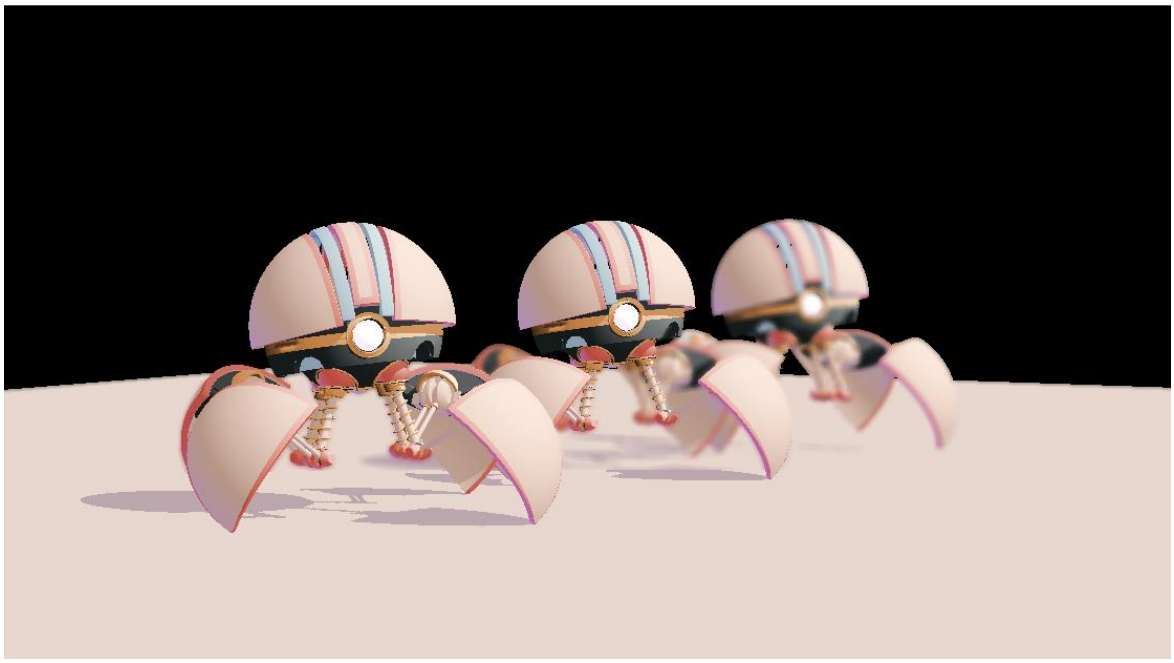

Figure 20. Example image for the Adaptive Bilateral Filter algorithm.

The difficulty in achieving this example image varied. Two of the testers rated it as being easy and two as normal, while one person did not achieve the result (See *Figure 21*).

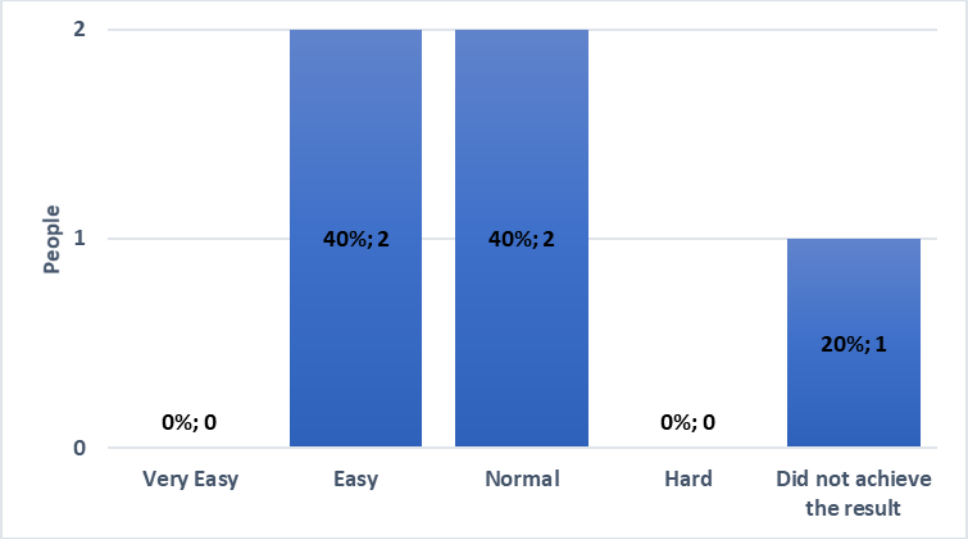

Figure 21. Adaptive Bilateral Filter difficulty scale.

The complexity of the parameters is seen on *Figure 22*. One person rated them as being very easy. This can be related to the fact that this person had previous knowledge of how a camera lens works and what the parameters need to do. Three testers rated the parameters as hard. It can be deduced that without prior knowledge of how a camera imaging process works the parameters would be hard to understand. The people using this algorithm would need prior reading materials on what the parameters do and how a blur is achieved in a camera.

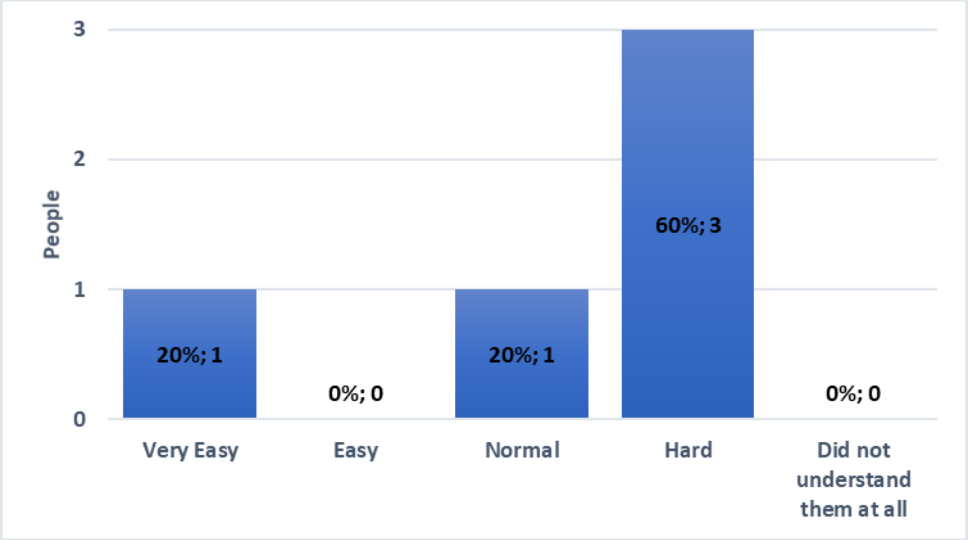

Figure 22. Parameter complexity for the Adaptive Bilateral Filter.

The testers did note major artefacts with the algorithm. The most obvious was the sharp cut-off from focus to blurred area. The answers also gave feedback on the general quality of the blur. For example, on higher blur scales the edges stay sharp and the blur looks squared and boxy while also black stripes and horizontal lines are noted. Around the focused

bots a small halo was noted. A bright blue box was seen on the bots as well. This was the same spot as seen in the naive approach in chapter 4.1.

General comments about the algorithm were that some parameters appear to be doing the same thing while `FocalLength` and `sharpeningFactor` seem to give a reversed effect. One tester noted that it would be better if all the parameters were under one macro node to simplify the workflow.

Polygonal Apertures Depth of Field Shader

The second algorithm the testers tried was the Polygonal Apertures DoF that is described in chapter 4.3. The example image is seen on *Figure 23*. The task was to try to tweak the parameters in a way that the first bot would be focused while the other two were blurred.

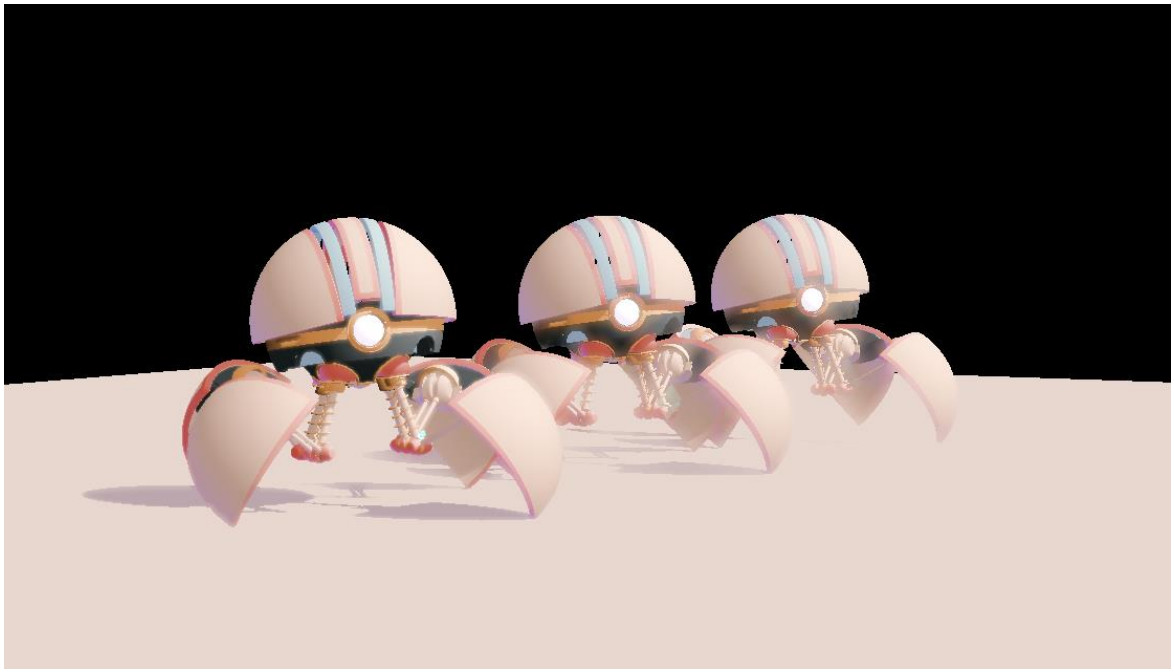

Figure 23. Polygonal apertures DoF example image.

Rating how difficult it was to achieve the same result varied (See *Figure 24*). One person was not able to achieve the result while other testers noted the difficulty from 1 to 3.

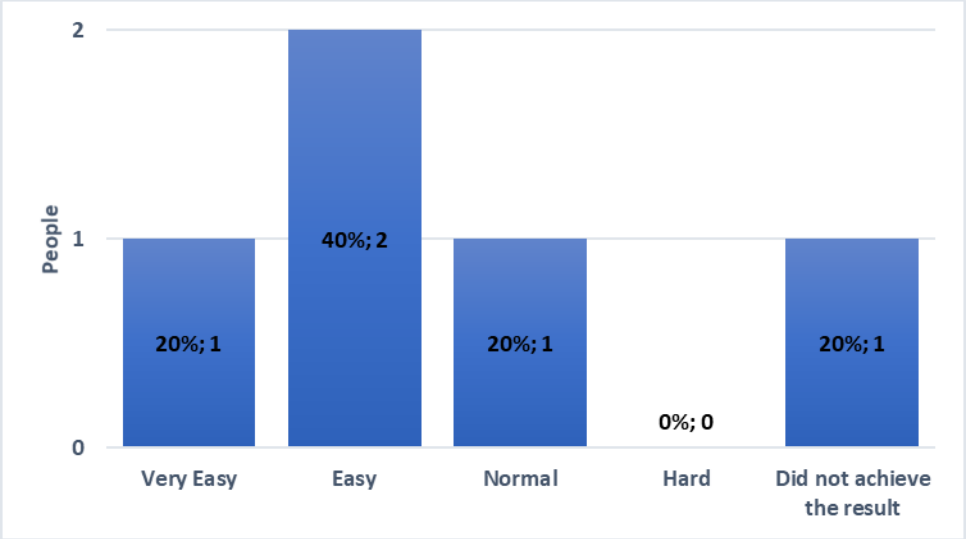

Figure 24. Polygonal Apertures DoF difficulty scale.

The complexity of the parameters is seen on *Figure 25*. It can be seen that for the majority the parameters were understandable, while one person did rate it as hard. Combining the information gathered from the previous answer and this question it can be inferred that tweaking the parameters together raises the difficulty level.

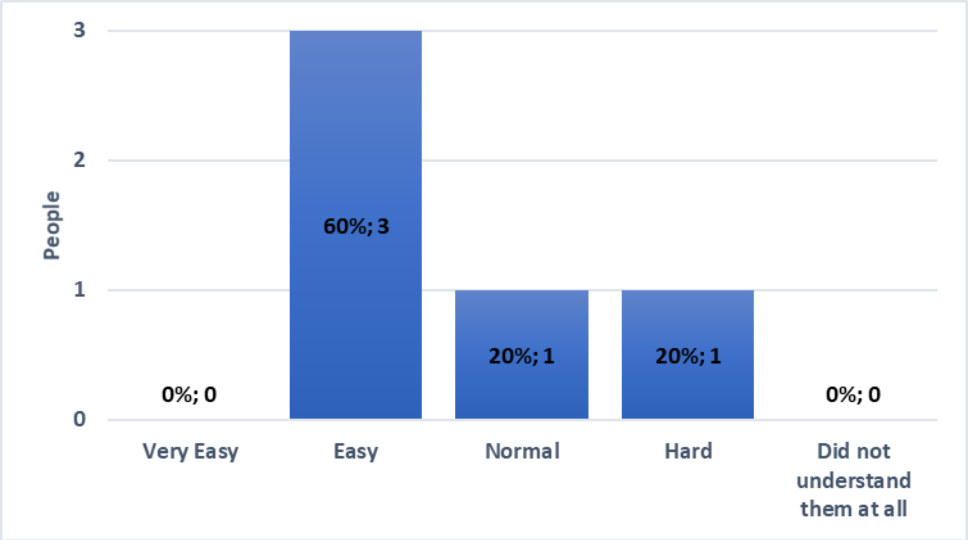

Figure 25. Polygonal Apertures DoF parameter complexity scale.

Some comments about the quality of the blur were the same as for the previous algorithm. The testers saw black stripes and bright misbehaving pixels (i.e. blue box). The blurred out areas still had sharp edges and there is a glowing effect with it as well. This glowing effect makes it unrealistic as one tester noted. Another remark was that when the `MaxCoCDiameter` exceeds a certain value, artefacts turn from rectangular pattern to a line pattern. This can be clamped down to a smaller value for better results. Also one tester noted

that there seemed to be a second in focus area near the camera. This could be related to the depth range clamping not starting from 0.

Adaptive Recursive Depth of Field Filter

The third algorithm that was tested was the Adaptive Recursive Filter. The example image is seen in *Figure 26*. The testers had to make it look that the last bot was in focus only.

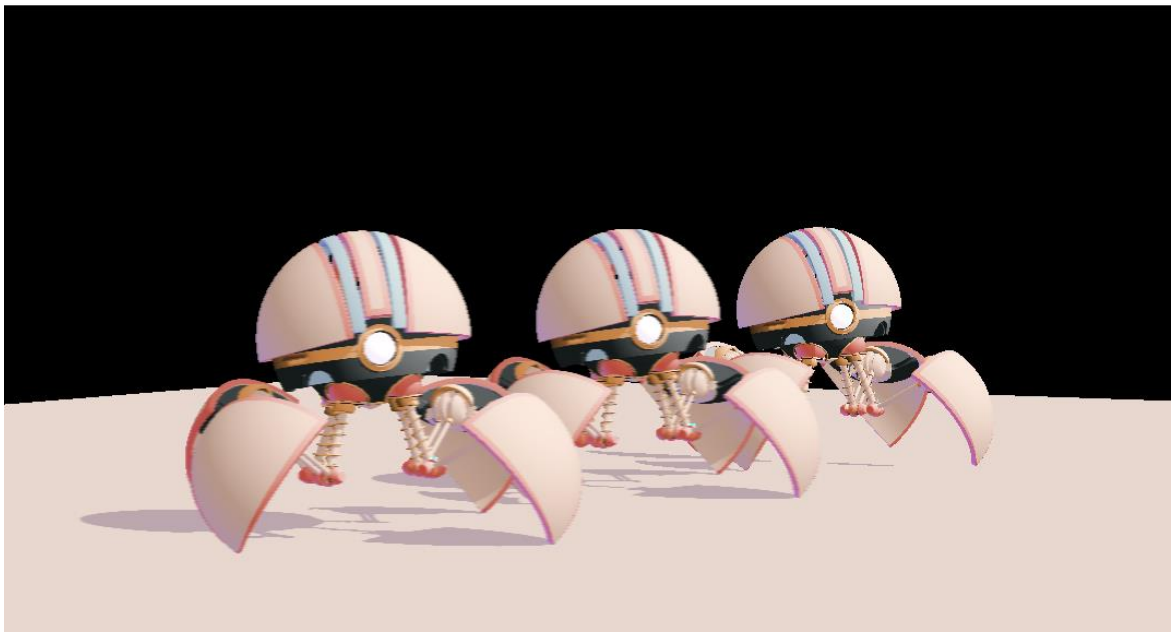

Figure 26. The example image of the Adaptive Recursive Depth of Field Filter.

However, this was rated by the testers as the most difficult. One person was not able to achieve the result while three people rated it as Hard (See *Figure 27*).

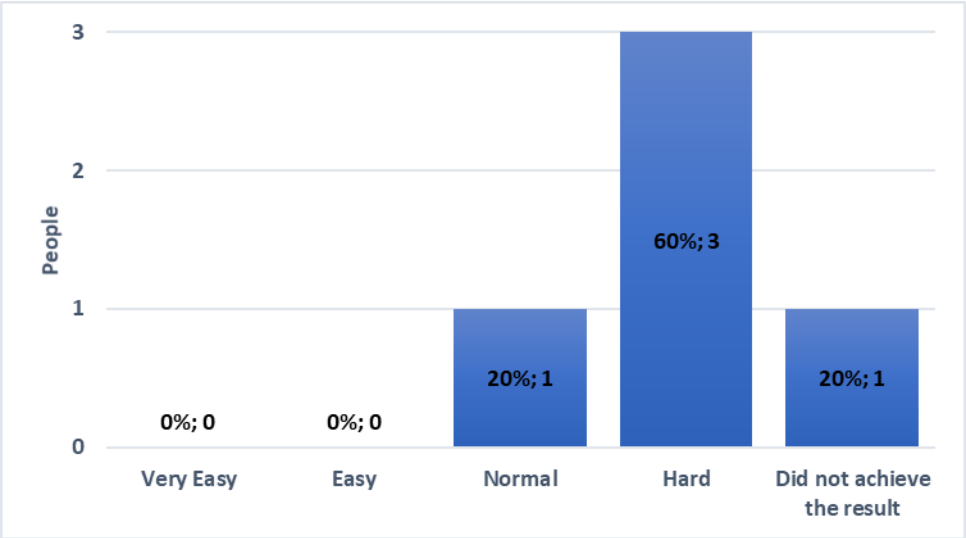

Figure 27. Adaptive Recursive Depth of Field Filter difficulty scale.

This has a correlation with the complexity of the parameters (See *Figure 28*). Four people rated them as being Hard and one person did not understand them at all.

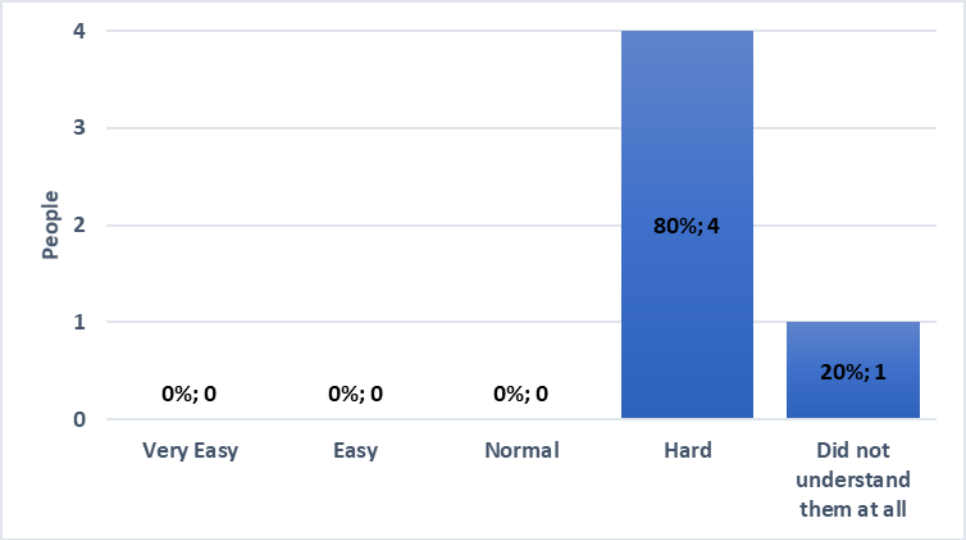

Figure 28. Adaptive Recursive Depth of Field Filter parameter complexity scale.

The testers noted that it produced little to no blur and that it was not easy to see a difference when parameters were changed. Taking into consideration the information gathered from *Figure 27* and *Figure 28*, it can be concluded that the small blur was the reason why the complexity of the parameters and the overall difficulty was so hard. One of the testers wrote that the implementation might not function properly.

Circular Separable Convolution Depth of Field

The fourth algorithm was rated by the testers as the easiest of the four. The test case example image is seen in *Figure 29*. This algorithm had two regions, the focus area and background. As such the testers had to make it look that the last bot was blurred and the first two in focus.

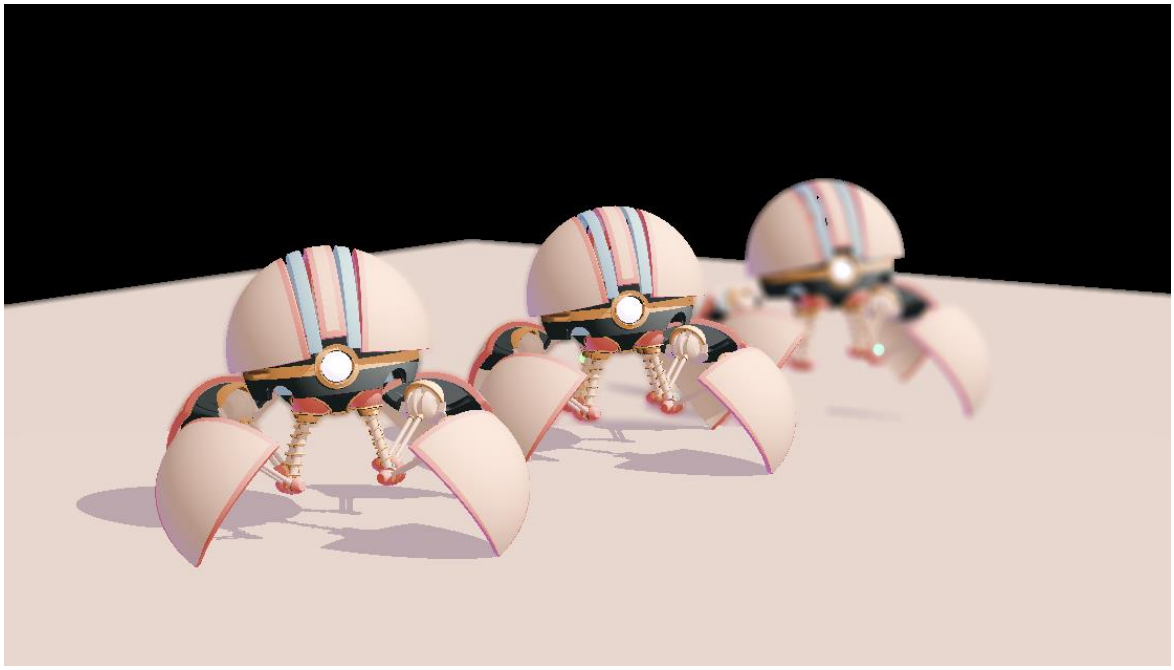

Figure 29. Example image for the Circular Separable Convolution DoF.

Three people marked the task as very easy while two people rated it as being easy (See *Figure 30*)

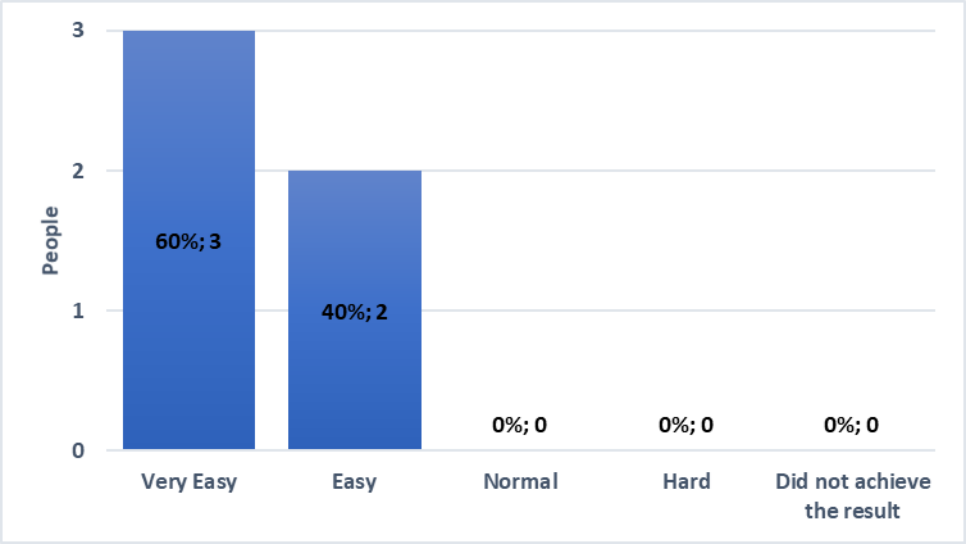

Figure 30. Circular Separable Convolution Depth of Field difficulty scale.

As the difficulty for the majority was easy, the same is for the complexity of parameters (See *Figure 31*). Since this implementation had three parameters in total, it was not hard to understand their function.

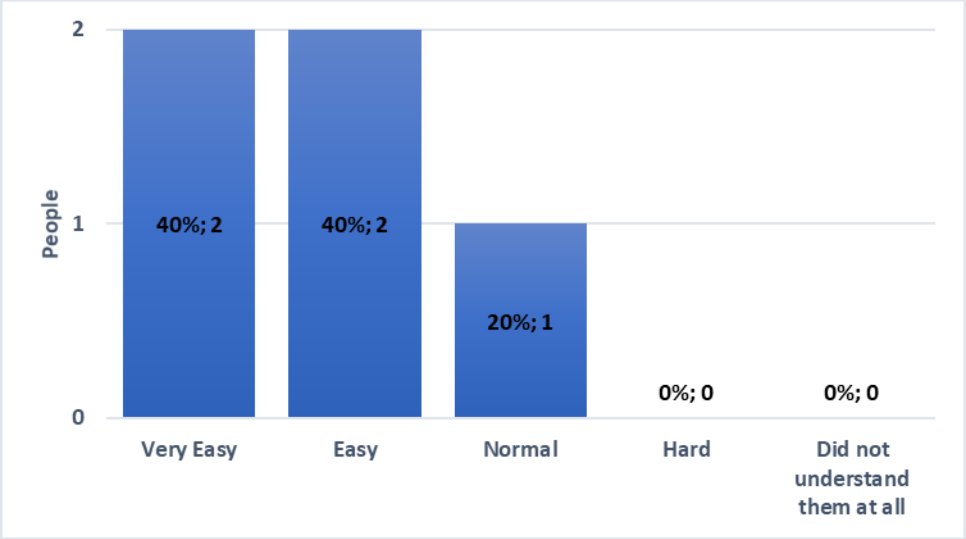

Figure 31. Circular Separable Convolution Depth of Field parameter complexity scale.

A major bug that the testers noted was that reducing the `KernelSize` parameter below 5 turned the area much brighter. This means that the `KernelSize` should have a minimum range of 5. Another noticeable artefact was that this algorithm had the same sharp cut-off as the Adaptive Bilateral Filter. It was stated in chapter 4.5 that the blending function was not known for the segmentation and thus this produced the sharp edge. However, the testers did remark that the quality of the blur was the best of the four and that bright pixels spread out naturally. It can be concluded that with finding the proper blending functions, this algorithm could have the most potential usage out of the four.

6.3 Stylized Extension Usability Testing

The stylized extension usability testing task gave the tester a lot freedom in making a DoF effect of their liking. The questions and general feedback asked were:

1. Would you use this effect in a scene of your own?
2. Please describe your decision for the last question.
3. Were there any artefacts? If yes, please describe them.
4. Rate how accurately did you manage to achieve the result you intended.
5. Any additional feedback or suggestions you can write here.

The second, third and fifth questions were text paragraphs. While the first question was a choice of two possible answers (“Yes” or “No”) and the fourth question a scale of 1 to 5 with 1 being “Exactly what I wanted” and 5 being “Was not possible”.

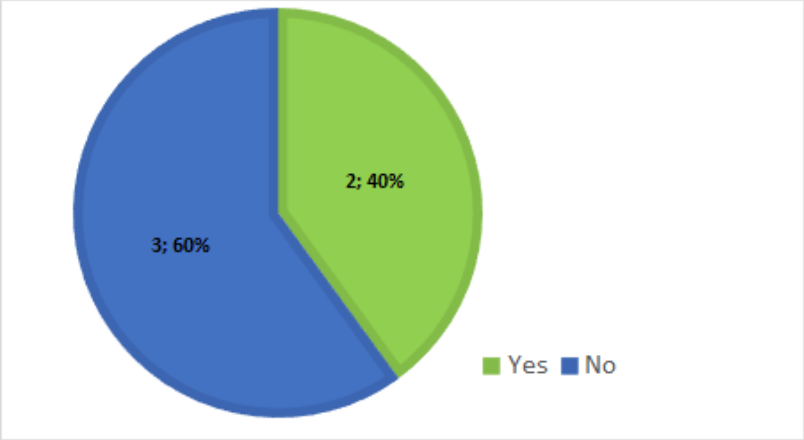

Figure 32. Testers responses if they would use the stylized extension.

For the first question two of the five testers responded that they would use it (See *Figure 32*). They described it as being a "wild" style and might be neat to use. However, the sharp transition between focus and background is restrictive. This is due to the stylization using the Circular Separable Convolution Depth of Fields algorithm for blurring.

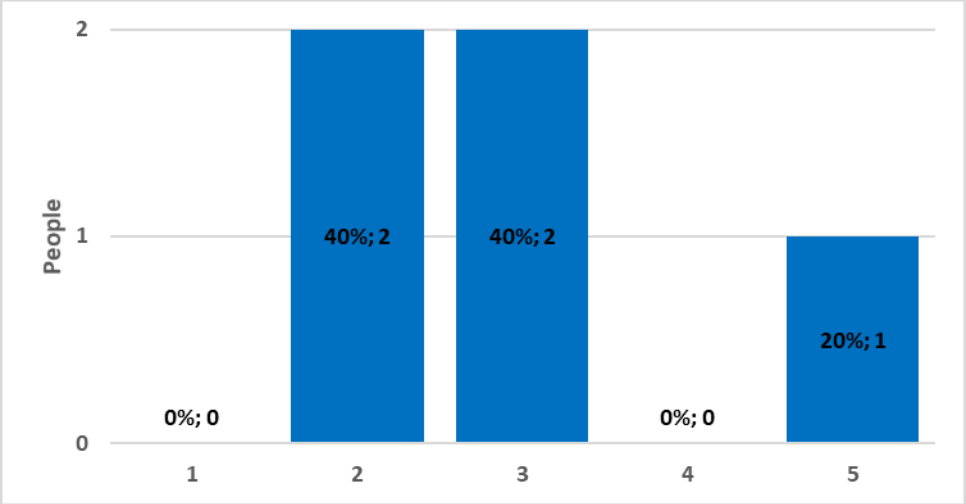

Figure 33. How accurately the testers achieved their intended result.

For the question of how accurately they achieved the intended result on a scale of 5, the answers showed that nobody achieved exactly what they had in mind (See *Figure 33*). Two people rated it with a 2 and two people with a 3. One person was not able to achieve their intended result at all.

The general feedback answers gave valuable improvement ideas. Changing the light direction was suggested to be made as a slider instead of a `vec3` coordinate. It would make the workflow simpler. In addition, it was remarked that although the warp idea is cool, then it might be better to infuse the warp with depth information to make it look like it belongs to the world. One tester wrote that they would like to remove the blur completely. In

addition, they would like to apply the blur after the toon shader in a way that it has discrete colours and that the character would look blobby. They also added that a further development would be where the background would be completely toon shaded with no gradients in the blurred region.

Taking into account all of the answers provided the stylized extension has room for further developments. These improvements would give artists a lot more customization options. With the help of the feedback received and developing this stylization further, it could have potential users in Flair.

7 Conclusion

In collaboration with Artineering a goal was set to implement four Depth of Field algorithms in Flair. These algorithms were chosen by a specific category, real-time rendering. Their implementations and quality were tested with potential future users of Flair and the DoF effects. After the implementations were complete one of these algorithms was chosen as a further improvement to make it a stylized Depth of Field effect. This stylization was tested with people from the computer graphics industry in conjunction with the four DoF algorithms.

The testing phase included five testers who have proficiency in computer graphics. The testing revealed that the algorithms did have artefacts. It was proposed that the Adaptive Recursive Depth of Field filter could have had implementation problems. This can be the issue and needs further evaluation to see if it is an implementation problem or the algorithms side product of having a small blur. Based on the testers feedback the Circular Separable Convolution Depth of Field algorithm produced the best aesthetic results. This could be further improved with a smooth transition of the blur and the focus area. Thus this algorithm is the most promising of the four to be used in Flair.

For the final stylization effect the toon shader could have different colouring options and the light direction would be better as a slider as is described chapter 6.3. Another idea was proposed that the warp could account for depth, so that it would look more natural in the scene. Also a future improvement should be to find how to properly blend the background edge onto the focus area. Taking into account the feedback received there is room for improvement.

Special thanks to the people who participated in the testing phase. With their help a lot of the artefacts and bugs were discovered. Thanks to their proficiency in computer graphics valuable feedback was received on how to improve the stylized extension.

References

[1] Tomas Akenine Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michal Iwanicki, Sébastien Hillaire, “Real-Time Rendering”, Fourth Edition, Taylor & Francis Group, 2018, p 527.

[2] Rafael C. Gonzales, Richard E. Woods. 2010. Digital Image Processing Third Edition. Pearson Education International. p 168-177.

[3] Wojciech Sterna. 2017. Practical Gather-based Bokeh Depth of Field. Wolfgang Engel. GPU ZEN 01 Advanced Rendering Techinques. Black Cat publishing. p 217-237.

[4] David C. Schedl, Michael Wimmer. 2013. Simulating Partial Occlusion in Post-Processing Depth-of-Field Methods. p187-200. GPU Pro 4 Advanced Rendering Techniques. Taylor & Francis Group.

[5] Wu S., Yu K., Sheng B., Huang F., Gao F., Ma L. (2012) Accurate Depth-of-Field Rendering Using Adaptive Bilateral Depth Filtering. In: Hu SM., Martin R.R. (eds) Computational Visual Media. CVM 2012. Lecture Notes in Computer Science, vol 7633. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-34263-9_33

[6] McIntosh, L. & Riecke, Bernhard & Dipaola, Steve. (2012). Efficiently Simulating the Bokeh of Polygonal Apertures in a Post-Process Depth of Field Shader. Computer Graphics Forum. 31. 1810-1822. https://www.researchgate.net/publication/261860589_Efficiently_Simulating_the_Bokeh_of_Polygonal_Apertures_in_a_Post-Process_Depth_of_Field_Shader

[7] Shibiao Xu, Xing Mei, Weiming Dong, Xun Sun, Xukun Shen, and Xiaopeng Zhang. 2014. Depth of field rendering via adaptive recursive filtering. In SIGGRAPH Asia 2014 Technical Briefs (SA '14). Association for Computing Machinery, New York, NY, USA, Article 16, 1–4. https://doi.org/10.1145/2669024.2669034

[8] Kleber Garcia. 2017. Circular separable convolution depth of field. In ACM SIGGRAPH 2017 Talks (SIGGRAPH '17). Association for Computing Machinery, New York, NY, USA, Article 16, 1–2. https://doi.org/10.1145/3084363.3085022

Appendix

I. Repository

https://github.com/siimanderson/DepthofFieldShaders

II. Glossary

Render – image synthetization.

Aperture – camera opening which lets light through.

Lens – optical device through which light rays refract.

Focal point – point where light rays converge for focused object.

Focal length – the focal points distance from the lens.

Circle of Confusion – an area that forms when light rays do not hit the focal point.

Bokeh – visual representation of the Circle of Confusion (e.g. circle or a polygon).

Kernel – a small matrix that is used in image processing, e.g. blurring.

Scene – view of an environment that contains 3D and 2D objects.

Shader – user defined program that is run on the GPU and has a specific task.

Object-space – coordinate system in which a specific 3D object is defined.

Downsampling – downscaling of a digital image.

Box blur – blurring operation where equally weighted kernel is used.

Weights – kernels values.

Artefact – an image distortion that happens due to image-processing.

Pixel bleeding – The colour of a pixel is leaking to the pixel of the background pixel.

Texture – an image that is mapped on to a 3D object.

Sampling – the act of getting a single value out of a greater population.

Normals – define the orienation towards the light source.

Normals texture – normals are stored in a texture where the RGB components are the XYZ coordinates of a normal.

License

Non-exclusive licence to reproduce thesis and make thesis public

1. I, Siim Anderson herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, Stylized 3D Depth of Field supervised by Raimond-Hendrik Tunnel and Santiago Montesdeoca.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Siim Anderson

07/05/2021