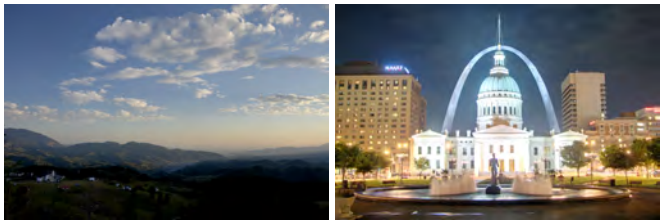
Adding Bloom to High-Dynamic-Range Tone Mapping

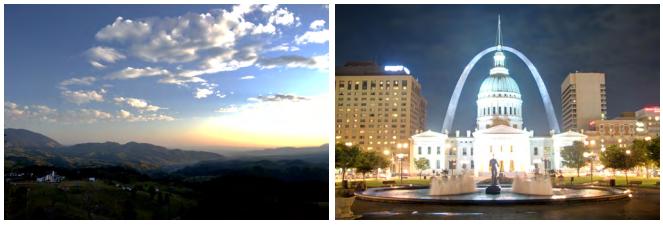
Ricardo Vogel
Supervisors: Ruben Wiersma , Elmar Eisemann
Delft University of Technology

Abstract

We present a technique for enhancing high-dynamic-range tone mapping algorithms by adding the bloom effect to bright areas. Bloom is based on the fact that real-life lenses convolve light and make bright areas emit a glow. The algorithm takes a set of images with different exposures as input, and performs a tone mapping algorithm on these. It then takes the image with the lowest exposure value to create the bloom effect. It then perform a convolution on this image with with a kernel that represents the response to one point of light. The resulting image is then added on top of the tone mapped image. We also present parameters to change the spread of the glowing effect, how bright an area needs to be to get a significant glow, and the intensity of the glow when applied. Furthermore, the kernel can be changed to create different types of glow and highlights. These things make the technique versatile and allows the photographer to customize the effect.



(a) Tone mapped images.



(b) Images as generated by the algorithm described in this paper.

Figure 1: Three different types of picture for two scenes. The images in (c) are the ones generated by the techniques presented in this paper. Left image by Daniel Pîrcălăboiu [3], right image by Kevin McCoy [4].

1 Introduction

When taking pictures in places with large differences in brightness, it is often not possible to capture the full dynamic range of the scene. To approximate the appearance of the high-dynamic-range scene, images with multiple exposure rates can be taken and combined using a technique called tone mapping. The resulting image can then be shown on a medium with a lower dynamic range, and has details in both dark and bright areas. Two examples of tone mapped images can be seen in Figure 1 (a).

The brightest areas in tone mapped images however, might not seem as bright as the photographer wants. One way bright areas in an image can be perceived brighter is by the presence of glare effects. These effects are caused by light scattering in the human eye or a camera lens and make bright points emit a glow [1, 2]. These effects can also be added to images to amplify the effects of bright areas.

One common way to achieve this glowing effect is the bloom shader effect. Bloom is used in video games and other digital media to create the illusion of a higher dynamic range, by making bright areas emit a glow [5]. Similarly to photography, digital media runs into the issue of having to display a high-dynamic-range scene onto a lower dynamic range media. The bloom effect helps solve this issue by creating a glow around the areas brighter than the displayable dynamic range. Outside of digital media, bloom is used as well, these however often rely on a more manual approach, or use a single image as input.

In this paper, we present an algorithm that adds the bloom effect to tone mapped high-dynamic-range images. The presented technique takes the images of multiple exposure rates as input. Using this additional information on the brightness of the scene, the brightest areas are selected to perform bloom on. We also use a modern form of bloom called convolution bloom, which is used by Unreal Engine [6], to make the bloom effect more realistic and versatile compared to older techniques. The generated bloom overlay is then merged with a tone mapped image to create an image that has the qualities of a tone mapped image, and has the intended glowing effect.

The images in Figure 1 (b) are generated using the technique presented in this paper.

One other major advantage of using this technique is the control the photographer has when editing the image. Multiple parameters to tweak the final result are presented, which can be used to increase the spread of the glow effect, select how bright an area has to be for it to give off a significant amount of bloom, and change the intensity of the bloom when applied. Furthermore, changing the kernel image used in the convolution bloom also greatly impacts the final result. Overall the technique is versatile and allows a photographer to change the qualities of the bright areas of the image.

2 Related Works

Because of the nature of the technique proposed in this paper, the related works is split into the two main parts of the algorithm. In Section 2.1 different tone mapping algorithms are discussed. Section 2.2 explains the theoretical basis for bloom, a common bloom implementation, and recent developments in the field.

2.1 High-dynamic-range Tone Mapping

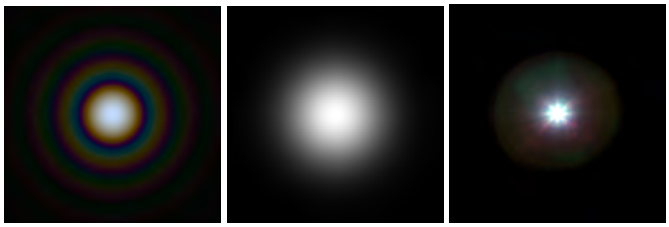
High-dynamic-range tone mapping operators can be split up into two categories, based on the methods they use to reduce contrast. Global operators apply the same mapping function across the whole image, while in local operators the mapping varies based on the neighborhood of a pixel.

A good example of a global operator is that created by Drago et al. [7]. They use a logarithmic compression of luminance values, which imitates the response to light of the human eye. The operator also provides several parameters to change the brightness, contrast compression, and detail reproduction in an intuitive way.

One major issue with local operators is that they can produce artifacts around hard edges if no way is found to circumvent them. The operator proposed by Fattal et al. uses gradients to detect the edges [8]. They argue that hard edges will create a large gradient, and finer textures correspond to smaller gradients. They try to find these large gradients and weaken them, while keeping smaller gradients intact. Similarly to the operator created by Drago et al. discussed earlier, this technique also uses the logarithm of luminance values. Durand and Dorsey use the edge-preserving bilateral filtering to separate a base layer and a detail layer, followed by reducing the contrast only in the base layer, assuring the detail is retained [9].

Mertens et al. [10] propose a solution that skips the assembly steps and simplifies the process. They assign a quality value to each pixel based on measures like contrast and saturation and then blend the pictures of different exposures based on the weighted average of these quality measures. To improve the quality of the result, a pyramid of different resolutions is used during the blending of the images which creates smoother transitions between different areas of the picture.

By its nature, tone mapping attempts to create an image where each area is as detailed as possible. Images like this might not always be what a photographer wants, so the technique is not always appropriate. This paper aims to add an extra layer on top of existing tone mapping techniques that uses the information from the pictures with multiple exposures to generate a glowing effect.

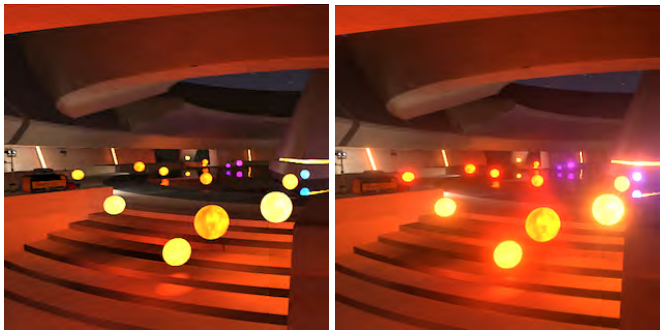


(a) A computer-generated Airy disk from diffracted white light. By SiriusB [12].

(b) Sampled Gaussian kernel.

(c) Example convolution kernel, by Epic Games Inc. [6].

Figure 2: Three different kernels. The Airy disk of (a) is the theoretical framework for bloom. The Gaussian kernel of (b) is a rough approximation that is often used. The kernel in (c) is an example kernel used by Unreal Engine to generate more appealing bloom.



(a) A scene without bloom.

(b) The same scene with the bloom effect added.

Figure 3: Two scenes created in the Unity engine showcasing the effects of bloom. By Unity Technologies [13].

2.2 Bloom

When bright lights hit a lens, such as one in a camera or in the human eye, it reflects in a pattern named the Airy disk [11]. This pattern has a large peak in the center with rings of light around it, as shown in Figure 2 (a). Although this effect happens for any point of light, it is especially noticeable in brighter areas. Bright areas look like they bleed into areas around it [1]. Effects like these make humans perceive the areas as brighter.

One method to add these glowing effects is the bloom shader effect. This is an effect often used in digital media such as animation and video games. An example of the effects of bloom can be seen in Figure 3.

One common bloom implementation is a simple Gaussian blur. The Gaussian blur is a decent and fast approximation of

the Airy disk. Its main downside is that it is missing the rings, as seen in 2 (b). In this version of bloom, a Gaussian blur is performed on the brightest areas and the result is added on top of the original image. This way, these areas appear to light up. While this technique often works well for video games, it is only an imprecise approximation. For this reason it might not look realistic enough for photography purposes.

A more modern technique, used by Unreal Engine, is called convolution bloom [6]. With this technique, you can set a kernel representing the response of the camera lens to a single point of light. An example of such a kernel can be seen in Figure 2 (c). Each pixel in the image contributes some of its brightness to its neighbors based on this kernel. The brighter the pixel is, the more brightness its neighbors will receive. Unreal's convolution bloom uses a fast Fourier transform to perform the convolution efficiently [14]. With the right kernel, this technique is an excellent implementation of the Airy disk theory, and as such it creates a more realistic result than the Gaussian bloom discussed earlier. It also allows for more customization compared to the Guassian bloom, since the kernel can be altered to create different types of effects.

For computer generated scenes, a threshold is often applied. Both Unreal Engine and the Unity engine allow for the use of such a threshold [6, 13]. This threshold can be set to 1, meaning maximum brightness, if the image has a high-dynamic-range. This way, only areas brighter than can be displayed will create the bloom effect. The threshold can also be disabled.

The examples so far have been for computer generated scenes. However, the bloom effect can also be used for photography. The *Oneric* plugin for Adobe Photoshop CC can be used to create bloom manually [15], and there are a number of tutorials online that explain how to achieve the effect, such as [16, 17].

The existing techniques for photographs rely on a single image as input, and usually use a Gaussian blur to create the effect. We aim to use the information provided by the multiple exposures to create a better effect, and to implement the convolution bloom to make a more realistic effect and to give more control over the final result.

3 Method

We aim to create an image that has the qualities of a tone mapped image, and has a glowing effect around the brightest areas to make them seem more bright. Furthermore, we aim to create a versatile technique that allows for customization of the effects.

Our technique first creates a tone mapped image using an existing tone mapping operator, followed by creating a bloom overlay using information from the lowest exposure rate. These two images are then blended to create a final result. We will assume the input images are perfectly aligned.Figure 4 shows an overview of the full method.

First the tone mapping algorithm is explained in Section 3.1, then the bloom overlay is discussed in Section 3.2. Section 3.3 explains how these two steps are merged, and Section 3.4 shows some important implementation details.

3.1 Tone Mapping

As discussed in Section 2.1, there are many different tone mapping operators. We need an operator that is high quality, efficient, and has some creative freedom. Mertens et al.' exposure fusion [10] provides these qualities. This operator assigns a weight to each pixel of each input image, and then averages them based on these weights to generate a final result. To blend them, a Laplacian pyramid is used for the images, and a Gaussian pyramid for the weights. The weights and images in the pyramid are multiplied on each level. The final result is obtained by collapsing the pyramid. This is done by keeping an intermediate result, set to the smallest image in the pyramid at first, and continuously scaling it up and adding the next image in the pyramid. One other reason this operator was originally chosen over other high-quality operators was to make use of the weighted blending for adding the bloom effect. This idea however did not prove useful in the end.

The final weight assigned to each pixel is based on three factors: the pixel's contrast, its saturation, and its well-exposedness. The specific method used to find the values of these factors is described by Mertens et al. [10]. These components each have a certain weight of their own. These weights are applied globally, and are not tied to specific pixels. Increasing one of these will increase the final weight of pixels with a certain quality. The final weight is calculated by taking the product of the factor, each raised to the exponent of their corresponding weight, as shown in Equation 1 (which is a simplified version of Mertens et al.'s original equation [10]).

$$W = C^{\omega_C} \times S^{\omega_S} \times E^{\omega_E} \tag{1}$$

In this equation, C, S, and E are the contrast, saturation, and well-exposedness measures. ω_C, ω_S, and ω_E are their corresponding weights. While these weights do not tend to drastically change the final result, their inclusion does add a small amount of extra creative control.

While this operator has some extra benefits, replacing it with another high-quality and efficient tone mapping operators will likely produce similar results.

3.2 Creating the Bloom Overlay

To create the bloom overlay, convolution bloom will be used. This technique is similar to that used by Unreal Engine [14].

To start, we select the input image with the lowest exposure value. This image has the most detail in the brightest areas. While it might seem more intuitive to select the image with the highest exposure value, given that this image looks the most like the desired effect, with the bright areas being very bright and bleeding into the surrounding area, the image with the lowest exposure value tells us which areas of the image are the brightest, and gives us the most information about these areas, allowing us to customize the bloom effect.

If we were to continue with this image directly, we have found that this creates too extreme of an effect around areas that have some brightness in the image, but are not among the brightest. To circumvent this issue, we use a smoothing

Figure 4: A complete overview of the tone map and bloom method. Given as input are pictures with multiple exposures, which are combined into a tone mapped image using Merten's Exposure fusion [10]. The picture with the lowest exposure rate is taken to create the bloom effect. It is smoothed first, then convolved with a kernel. The tone mapped image and the bloom overlay are combined into a final image. Original images by Kevin McCoy [4].

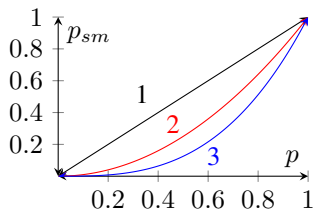
function. While a simple threshold, as used by Unreal Engine and the Unity engine [6, 13], would already help, this could create an undesired pattern in pixels close to the threshold. Instead, we use a function similar to that used in gamma correction to make darker areas less prevalent. The function for this is shown in Equation 2, with p_{sm} being the new value, p being the old value of the pixel, and σ being the *smoothing factor*. The equation is applied to each color channel of each pixel separately, with the color range being a decimal number between zero and one.

$$p_{sm} = p^{\sigma} \tag{2}$$

By using this equation, the brightest areas will remain bright, and darker areas will become darker without becoming black. The falloff rate is determined by the smoothing factor σ. A higher smoothing factor will assure that only the brightest areas will remain bright. A smoothing factor of one means there will be no smoothing, and a smoothing factor lower than one will not create the desired effect. Figure 5 shows the resulting images for three different smoothing factors, as well as a plot for the function.

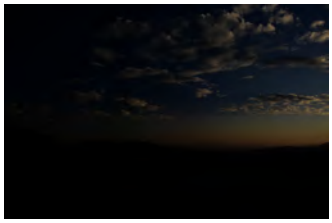
This function was chosen because it keeps pixels with the maximum and minimum possible brightness at the same value, while scaling the rest down. In a way it allows the user to select how bright an area must be to give off a significant amount of bloom. In Figure 5 (b) the whole sky will still give off a significant amount of bloom, while in figure (d) only the right part will seem significant, with the left part giving of a more subtle glow.

To perform the convolution bloom, an appropriate kernel should be provided. This is perhaps the most impactful parameter of the algorithm, as the kernel can change the image drastically. To create the most realistic effect, the kernel should simulate how a camera or the human eye captures a single point of light in the center, similar to the Airy disk shown in Figure 2 (a). Besides the choice of kernel, the kernel structure should not be cropped too tightly, since a too tight crop will create a bloom overlay with square edges. An

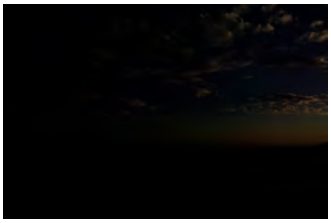


(a) A graph showing the values of pixels after smoothing.

(b) Image where $\sigma = 1$ (i.e. no smoothing).



(c) Image where $\sigma = 2$.



(d) Image where $\sigma = 3$.

Figure 5: A comparison of three different smoothing factors. Pixels with a brightness of one and zero stay like that, while other pixels get darker. The smoothing factors determines the falloff rate. A smoothing factor of one means no smoothing is applied. Original images by Daniel Pîrcălăboiu [3].

example of an appropriate kernel can be seen in Figure 2 (c).

Before the convolution is performed, the kernel can be resized with a given x and y size. This allows for the spread of the bloom to be altered easily without the help of outside tools.

To perform the convolution, the values in the kernel should be in the range of zero to one. Effectively, the kernel acts as a multiplier. Each color channel in the image is convolved separately with its corresponding channel in the kernel. Each pixel is added to its local area weighted by the kernel. The higher the value of pixel, the more it influences its neighbors. A more formal definition can be found in Equation 3, with

$g(x, y)$ being the new image, $f(x, y)$ being the old picture, ω being the kernel, and a and b being the width and height of the kernel image.

$$g(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx, dy) f(x + dx, y + dy) \quad (3)$$

Given the versatility of the convolution bloom method, different effects can be achieved by changing the kernel. By using a colored kernel for example, the brightest areas can give off a colored glow. The shape of the kernel can also be seen in smaller highlights. Another interesting effect is a more horizontal kernel, which creates horizontal glows similar to those created by anamorphic lenses.

The Gaussian bloom alternative, as mentioned in Section 2.2, is also still possible within convolution bloom. A Gaussian blur is simply a convolution using a Gaussian function as kernel. In practice, the Gaussian function is sampled to create a kernel. A sampled kernel such as the one in Figure 2 (b) can be provided to the algorithm.

3.3 Blending the Components

The final step in the process is to blend the tone mapped image and bloom overlay. One final parameter is introduced, the bloom intensity α. The new image is computed as shown in Equation 4, again with each pixel and color channel separately. p_f represents the final computed pixel, p_b the pixel in the bloom overlay, and p_t the pixel in the tone mapped image. The bloom intensity should be positive for the desired bloom effect.

$$p_f = \alpha * p_b + p_t \quad (4)$$

3.4 Implementation Details

The most important implementation detail is the use of the fast Fourier transform instead of the standard convolution defined in Equation 3. This will greatly increase the efficiency of the generation of the bloom overlay. The resulting bloom overlay should also be the same size as the input image to allow for the merging step.

4 Results

In this section, the images created using the technique described in Section 3 are shown. We also show how the different parameters change the image. Appendix A shows the settings used for each figure.

Adding the bloom overlay has an impact on different kinds of high-dynamic-range images. Take for example Figure 6. In this image the bloom effect highlights the areas close to the horizon, and makes the difference between the blue and orange parts of sky more prevalent. Compare this to Figure 7, a picture taken in a city at night. While the tone mapped image already has some glow coming off bright areas, adding the bloom overlay helps highlight these areas even more.

(a) Tone mapped

(b) Bloom overlay

(c) Final result

Figure 6: The three steps of the algorithm, tone mapping, bloom, and the fused result, applied to a mountain scene. Original images by Daniel Pîrcălăboiu [3].

An example of a picture that does not work as well can be seen in Figure 8. The image with the lowest exposure value, (b), is very dark and primarily green. Setting the kernel size to be small, like in (c), works well enough, but when setting the size to be larger, like in (d), the window emits a green glow that looks unappealing and unrealistic.

Changing the kernel is one of the most impactful changes one can make to change the outcome. Figure 9 shows the same image with different kernels. The kernels can be seen in Figure 10. The first image shows a very wide but thin kernel. This creates an effect similar to an anamorphic lens used in cinema. The second image uses a cross shape as a kernel, which can be seen in highlights. The third images uses a red kernel, which creates red highlights around red-tinted and bright areas. The final image uses three colored squares as a kernel, which creates a sort of "glitching" effect.

A comparison of different smoothing factors is found in Figure 11. The first image shows that the smoothing factor is an important step, without it some images turn out with too much bloom. The second image still has too much glow in the sky and railing of the bridge. From the third picture on, the results are as intended, with a different amount of glow.

A similar effect, but less extreme, can be seen when changing the bloom intensity α, as seen in Figure 12. This factor does does not change where the bloom is applied, but how bright the bloom is. In the first picture the bloom is still noticeable, but more subtle than in the fourth picture.

(a) Tone mapped

(b) Bloom overlay

(c) Final result

Figure 7: The three steps of the algorithm, tone mapping, bloom, and the fused result, applied to a picture of the Jefferson National Expansion Memorial. Original images by Kevin McCoy [4].

The performance of our Python implementation can be seen in Table 1. Overall, it seems that the tone mapping and bloom steps take a similar amount of time, with slight variations depending on the amount of different exposures. The merging step seems mostly negligible.

5 Discussion

The discussion is structured as follows: Section 5.1 discusses the creative control in the method, and how the different parameters affect it. In Section 5.2 our method will be compared to existing bloom methods, and finally Section 5.3 shows shortcomings of the algorithm and offers some suggestions for future research.

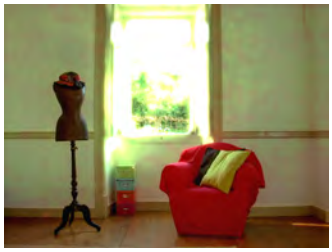
5.1 Artistic Control

Changing the kernel can have major effects on the outcome of the algorithm. Changing the shape creates highlights that look like the kernel. Looking at Figure 9 (b) and (c) show this well in small highlights such as the street lights. In generally bright areas, such as the entrance to the courthouse, the shape is less prevalent but the bloom is still appealing. One area where the shaped kernels do not work well is detailed bright areas, such as the sign on top of the building on the left side

(a) Tone mapped

(b) Lowest EV Image

(c) With a small kernel Size



(d) With a large kernel

Figure 8: A showcase of a scene where the algorithm does not work as well. Original images by Axel Jacobs [18],

w$\times h \times N$	K	TM	B	M	T
1440$\times$1080 $\times$ 5	55$\times$55	0.8	0.7	0.04	1.6
1440$\times$1080 $\times$ 3	55$\times$55	0.5	0.7	0.04	1.3
1440$\times$1080 $\times$ 1	55$\times$55	0.0	0.7	0.04	0.7
2800$\times$2112 $\times$ 4	55$\times$55	2.3	2.7	0.16	5.5
2800$\times$2112 $\times$ 4	5$\times$5	2.7	2.8	0.17	5.7
4928$\times$3264 $\times$ 3	55$\times$55	6.1	7.6	0.41	14.0

Table 1: Computation times for the technique. All times are in seconds. N is the amount of different exposures, K is the kernel size, TM is the tone mapping part, B is the bloom overlay generation, M is the merging step, and T is the final merged time. The average is taken over 100 runs of our Python implementation. The total time is the average of 100 runs of the whole algorithm, so might not be equal to the sum of the averages of all parts.

of the picture. With the more standard kernel, like in Figure 7, there is a lot of overlap between kernels applied around the bright areas, this way details are less noticeable.

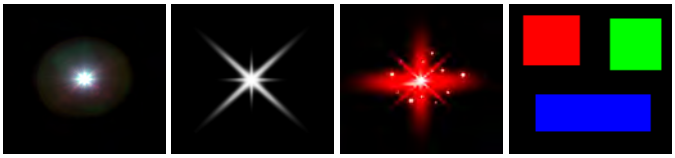
The other two major parameters are the smoothing factor σ and the bloom intensity α. Both of these parameters change the intensity of the bloom effect, but in different ways. The smoothing factor determines how bright areas need to be to have bloom applied, while the bloom intensity only makes areas with bloom more or less bright.

The smoothing factor seems to have a de facto minimum value which depends on the picture. The smoothing factors in Figure 11 (a) is clearly not high enough, since the bloom is applied even to areas that are not very bright. To a lesser extent the same can be said about (b). Once this minimum is found, the images are generally appealing, but do look different, allowing the photographer to choose what effect they want.

(a) Wide and thin kernel (b) Cross shape kernel

(c) Red star kernel (d) Colored kernel

Figure 9: Images with different kernels applied, showcasing the effect the kernel has on the final result. All other settings remain the same. Original images by Kevin McCoy [4].



(a) Kernel by Epic Games Inc. [6] (b) Cross kernel by Starline [19] (c) Red star kernel by Starline [20] (d) Multiple colors kernel

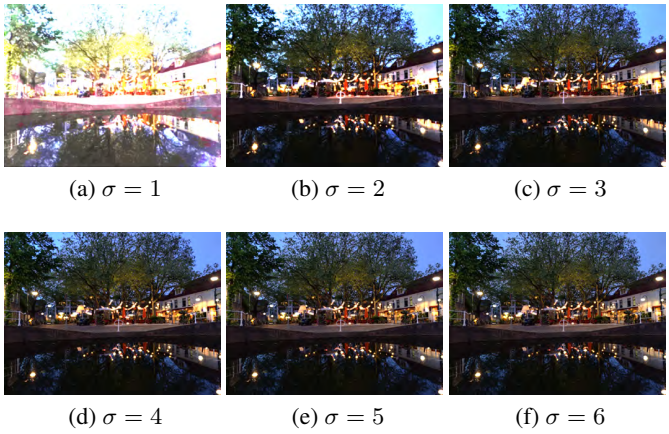
Figure 10: Different kernels used in the results.

The bloom intensity does not suffer from the same issue. All images in Figure 12 seem appealing, but there is a noticeable difference. This difference is especially noticeable on the right side, where the glow overlaps more with a higher bloom intensity. An intensity of one seems to be a good starting point for any image, from which the result can be tweaked.

5.2 Relation to Previous Works

The methods presented in this paper differ from those often used previously in several ways. We compare four of our additions, the use of the custom kernel, the use of the smoothing factor instead of a threshold, the use of a higher dynamic range, and the use of the image with the lowest EV, to alterations of our algorithm with these features missing.

Two common elements used by other techniques for generating bloom are the use of Guassian blur and a threshold. Figure 13 (c) shows a version of our method with the bloom overlay changed to be a Gaussian blur of the smoothed version of the lowest EV image. This glowing does look similar to our method, showing that the Guassian blur is a good alternative to the custom convolution kernel. However, there is not as much option for customization, since the effects shown in Figure 9 are not achievable. Another common tool that is used to create bloom is the threshold. Figure 13 (d) shows a version of our algorithm with the smoothing factor replaced with a threshold. This creates some artifacts, for example the unappealing red glow on the building on the right. This is caused by values close to the threshold being cut off.



(a) $\sigma = 1$ (b) $\sigma = 2$ (c) $\sigma = 3$

(d) $\sigma = 4$ (e) $\sigma = 5$ (f) $\sigma = 6$

Figure 11: A comparison of different smoothing factors. All other settings remain the same.

The other addition presented in this paper is the use of high-dynamic-range tone mapping instead of a regular image. Figure 13 (e) shows an image which only uses the medium exposure value to generate the result. The bloom effect is still present, but there are less details in the image. The reflections in the water are not as clear, and the colors of the trees are not as saturated. When using the tone mapped image to create the bloom overlay, as seen in figure Figure 13 (f), these details are present. With a higher smoothing factor, this method works for this scene. The same cannot be said about Figure 14. Even with an extremely high smoothing factor or a maximal threshold, the image looks unappealing. This is because in the tone mapped image many of the pixels have the maximum value within this dynamic range. This shows us that taking the image with the lowest exposure value does help the algorithm, since it gives information on pixels above the maximum value in the tone mapped image.

5.3 Shortcomings and Future Work

One result where the algorithm did not perform very well was Figure 8. This image has one prevalent color in the lowest EV image. This makes the glow too green if the kernel size is set too large. The ideal result in this image might be a white window, with glow around it. A good addition to the algorithm could be a pre-processing step where certain areas can be selected to “force” bloom.

One other potential issue with the technique is that the bloom is added without discrimination. This means that even bright areas that the photographer might not want highlighted will still emit a glow. The image in Figure 12 for example

(a) $\alpha = 0.5$ (b) $\alpha = 1.0$

(c) $\alpha = 1.5$ (d) $\alpha = 2.0$

Figure 12: A comparison of different intensities. All other settings remain the same.

(a) A tone mapped image. (b) An image generated using our method. (c) An image generated using Gaussian bloom.

(d) An image generated with a threshold. (e) An image without tone mapping, using the picture with an exposure value of zero. (f) An image which uses the tone mapped image to generate the overlay.

Figure 13: The method described in this paper compared to different alterations or existing techniques.

needed a high smoothing factor to ensure the sky did not get the bloom effect as well. Similarly to the previous problem, a solution for this could be a pre-processing step where certain areas can be selected to exclude from the bloom generation.

One other shortcoming of the algorithm is the supposed minimum value of the smoothing factor. Figure 11 (a) is clearly not a pleasing image, and the same holds for most other images when the smoothing factor is equal to one. Finding this minimum value currently has to be done by trial and

(a) An image generated using the methods described in this paper. (b) An image which uses the tone mapped to generate the bloom overlay.

Figure 14: A case where generating the bloom overlay on the the tone mapped image produces a bad result. Original images by Kevin McCoy [4].

error. Determining a decent smoothing factor automatically, to then be customized to the photographer's liking, could be an interesting addition to the algorithm.

The patterns around highly detailed bright areas are another point of improvement. It is possible that this is a a problem with the kernels used. Researching what types of kernels are good for convolution bloom could be a useful topic, not just to extend the algorithm presented in this paper, but also for other areas where bloom is applied, such as video games and computer animation.

With regards of the performance of the implementation, a 16 MP image with three different exposure rates took just over fourteen seconds. If the parameters for tone mapping are not changed, that part will only need to be computed the first time, reducing the total time for further runs to just under eight seconds. This is an acceptable time to wait to make the system interactive. Our implementation is unoptimized, and could likely be sped up, possibly by making use of a GPU-accelerated FFT implementation, such as [21].

6 Conclusions

We presented a technique for adding a glowing effect to tone mapped high-dynamic-range images. The technique is based on the fact that real-life lenses convolve light and make bright areas emit a glow. The algorithm takes a set of images with different exposures as input, and produces a tone mapped image using Mertens et al.'s Exposure Fusion [10]. The technique then creates an overlay using a modern form of bloom called convolution bloom. This overlay is generated by convolving a kernel with the image with the lowest exposure value. For a realistic effect, this kernel should be similar to a camera's response to a single point of light.

The custom kernel allows for a great deal of customization. Changing the kernel can greatly impact the look of the final result, and can create different types of effects. Furthermore, parameters are presented that add further control for the photographer, allowing them to chose how bright an area needs to be to emit a significant amount of glow, how intense the glow will be, and the spread of the glow. Overall, the technique is versatile and allows the photographer to customize the effect.

Broader Impact and Responsible Research

When creating software it is always important to consider its broader impact and ethical implications. While its intended use is often creating art, software that alters or creates media does bring ethical concerns. Modified images can be used for various illegal and otherwise unethical purposes, including misleading information and propaganda [22]. While these concerns are present in any image manipulation software, the technique presented in this paper only serve a specific purpose that might not be as susceptible to unethical manipulation.

One other important consideration for research is its reproducibility. If the paper is not reproducible, it is difficult for others to extend on the paper and perform further research. To help with this, the Python source code used to generate the results will be available on GitHub[1], released under the MIT license. Appendix A also shows a table of the settings used in the images in the paper. Finally, most pictures used in this paper are released under a Creative Commons license, and all pictures taken by us will also be released under the Creative Commons Attribution-ShareAlike 4.0 license. With the source code, settings, and example images easily available, others will be able to reproduce the results of this paper.

Acknowledgments

We would like to thank Ruben Wiersma and Elmar Eisemann for their support during the project. We would also like to thank the academic communication skills lecturers and the responsible research coaches of the TU Delft. Thanks also goes to Kevin McCoy and Axel Jacobs for sharing their images with a Creative Commons license, and to Daniel Pîrcălăboiu for allowing us to use his images.

References

[1] G. Spencer, P. Shirley, K. Zimmerman, and D. P. Greenberg, "Physically-based glare effects for digital images," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '95. New York, NY, USA: Association for Computing Machinery, Sep. 1995, pp. 325–334. [Online]. Available: http://doi.org/10.1145/218380.218466

[2] C. Beckman, O. Nilsson, and L.-E. Paulsson, "Intraocular light scattering in vision, artistic painting, and photography," *Applied Optics*, vol. 33, no. 21, pp. 4749–4753, Jul. 1994, publisher: Optical Society of America. [Online]. Available: http://www.osapublishing.org/ao/abstract.cfm?uri=ao-33-21-4749

[3] D. Pîrcălăboiu, 2013.

[4] K. McCoy, "The Jefferson National Expansion Memorial, including the Gateway Arch and Old Courthouse, in St Louis, MO, USA. Taken at night as part of a set of different exposures." May 2008. [Online]. Available: https://en.wikipedia.org/wiki/File:StLouisArchMultExpEV-4.72.JPG

[5] G. James and J. O'Rorke, "Real-Time Glow," May 2004. [Online]. Available: https://www.gamasutra.com/view/feature/130520/realtime_glow.php

[6] Epic Games, Inc., "Bloom | Unreal Engine Documentation." [Online]. Available: https://docs.unrealengine.com/en-US/RenderingAndGraphics/PostProcessEffects/Bloom/index.html

[7] F. Drago, K. Myszkowski, T. Annen, and N. Chiba, "Adaptive Logarithmic Mapping For Displaying High Contrast Scenes," *Computer Graphics Forum*, vol. 22, no. 3, pp. 419–426, 2003, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00689. [Online]. Available: http://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00689

[8] R. Fattal, D. Lischinski, and M. Werman, "Gradient domain high dynamic range compression," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '02. New York, NY, USA: Association for Computing Machinery, Jul. 2002, pp. 249–256. [Online]. Available: http://doi.org/10.1145/566570.566573

[9] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 257–266, Jul. 2002. [Online]. Available: https://dl.acm.org/doi/10.1145/566654.566574

[10] T. Mertens, J. Kautz, and F. Van Reeth, "Exposure Fusion," in *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, Oct. 2007, pp. 382–390, iSSN: 1550-4085.

[11] C. Sheppard, "MICROSCOPY | Overview," in *Encyclopedia of Modern Optics*. Elsevier, 2005, pp. 61–69. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B012369395000823X

[12] SiriusB, "Airy disk and pattern from diffracted white light (D65 spectrum). The color stimuli have been calculated in the CIE 1931 color space and then converted into sRGB. Apart from the sRGB definition there is a moderate additional gamma correction of 0.7 0.8 to enhance brightness in the outer rings. This may cause a slight but acceptable distortion in colours, however." Apr. 2018. [Online]. Available: https://en.wikipedia.org/wiki/File:Airy_disk_D65.png

[13] Unity Software Inc., "Unity - Manual: Bloom," 2017. [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/PostProcessing-Bloom.html

[14] Epic Games, Inc., "Unreal Engine - Image-Based (FFT) Convolution for Bloom," 2017. [Online]. Available: https://www.youtube.com/watch?v=SkJgopq-JQA

[15] Composite Nation, "Oniric - A powerful non-destructive glow generator for Adobe Photoshop CC 2018 and up." [Online]. Available: https://www.compositenation.com/plugins/oniric

[1] https://github.com/ricardovogel/tonemap-and-bloom

[16] 2000px Media Inc., "_USBloom | Photoshop Tutorials," Nov. 2006, section: Photo Effects. [Online]. Available: https://www.photoshoptutorials.ws/photoshop-tutorials/photo-effects/bloom/

[17] V. Kovalcik, "Add a Bloom Effect to Lights in Your Pictures," Jan. 2017. [Online]. Available: https://learn.zoner.com/add-a-bloom-effect-to-lights-in-your-pictures/

[18] A. Jacobs, "Open window with armchair and manequin. Sample scene for HDRI," Jan. 2006. [Online]. Available: https://en.wikipedia.org/wiki/File:HDRI_Sample_Scene_Window_-_01.jpg

[19] starline, "White sparkles and lens flare big set Free Vector," 2021. [Online]. Available: https://www.freepik.com/free-vector/white-sparkles-lens-flare-big-set_12686002.htm

[20] ——, "Set of transparent blue light streak and lens flares Free Vector," 2021. [Online]. Available: https://www.freepik.com/free-vector/set-transparent-blue-light-streak-lens-flares_12686004.htm

[21] Nvidia Corporation, "cuFFT," Jan. 2012. [Online]. Available: https://developer.nvidia.com/cufft

[22] M. J. Shapter, "Image manipulation and the question of ethics," *Journal of Audiovisual Media in Medicine*, vol. 16, no. 3, pp. 130–132, 1993, publisher: Taylor & Francis _eprint: https://doi.org/10.3109/17453059309064840. [Online]. Available: https://doi.org/10.3109/17453059309064840

A Image settings

Figure	Kernel	σ	α	Kernel size
6 & 1 (b)	10 (a)	4.5	1.0	25, 25
7 & 1 (b)	10 (a)	3.0	1.4	55, 55
8 (c)	10 (a)	1.5	1.0	25, 25
8 (d)	10 (a)	1.5	1.0	55, 55
9 (a)	10 (a)	3.0	1.4	155, 5
9 (b)	10 (b)	3.0	1.4	55, 55
9 (c)	10 (c)	3.0	1.4	55, 55
9 (d)	10 (d)	3.0	1.4	55, 55
11	10 (a)	*	1.0	25, 25
12	10 (a)	8.0	*	25, 25
13 (b)	10 (a)	3.0	1.0	15, 15
13 (c)	†	3.0	1.0	15, 15
13 (d)	10 (a)	‡	1.0	15, 15
13 (e)	10 (a)	7.0	1.0	15, 15
13 (f)	10 (a)	10.0	1.0	15, 15
14 (a)	10 (a)	3.0	1.4	55, 55
14 (b)	10 (a)	10000	1.4	55, 55

Table 2: Settings for different figures. *: These settings are variable, see the figures for their different values. †: This image uses OpenCV's Guassian blur instead of a convolution. ‡: This image uses a threshold of 100 instead of a smoothing factor.

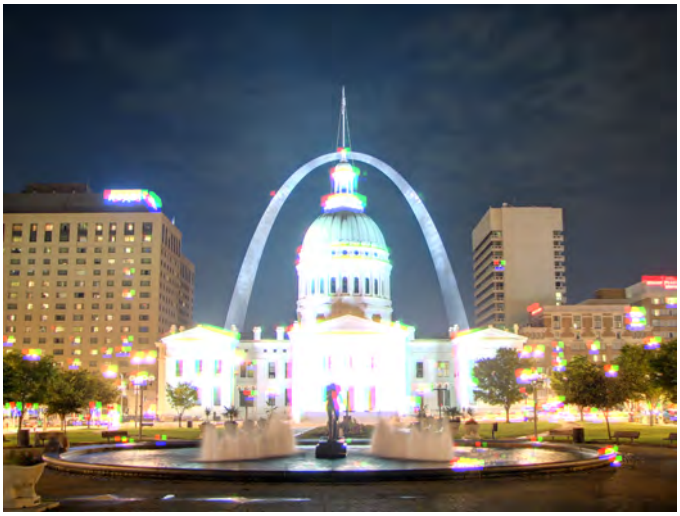
B Enlarged Figures

The following pages show larger versions of some figures from the main paper that might be to small when printed.

(a) Wide and thin kernel

(b) Cross shape kernel

(c) Red star kernel



(d) Colored kernel

Figure 9: Images with different kernels applied, showcasing the effect the kernel has on the final result. All other settings remain the same. Original images by Kevin McCoy [4].

(a) $\sigma = 1$ (b) $\sigma = 2$ (c) $\sigma = 3$

(d) $\sigma = 4$ (e) $\sigma = 5$ (f) $\sigma = 6$

Figure 11: A comparison of different smoothing factors. All other settings remain the same.

(a) $\alpha = 0.5$

(b) $\alpha = 1.0$

(c) $\alpha = 1.5$

(d) $\alpha = 2.0$

Figure 12: A comparison of different intensities. All other settings remain the same.

(a) A tone mapped image.

(b) An image generated using our method.

(c) An image generated using Gaussian bloom.

(d) An image generated with a threshold.

(e) An image without tone mapping, using the picture with an exposure value of zero.

(f) image which uses the tone mapped image to generate the overlay.

Figure 13: The method described in this paper compared to different alterations or existing techniques.

(a) An image generated using the methods described in this paper.

(b) An image which uses the tone mapped to generate the bloom overlay.

Figure 14: A case where generating the bloom overlay on the the tone mapped image produces a bad result. Original images by Kevin McCoy [4].