Abstract

Lens flare occurs in photography when rays from a very bright light have internal reflections and scatter in the lens system of a camera, overlaying the captured image with artifacts such as blown out starbursts, colored shapes, rainbow patterns and haze. While sometimes used for an artistic effect, lens flares can also be undesirable. In this project we aimed to train a deep learning network to automatically remove lens glare from images, predicting what would have been captured without internal reflections or scattering in the lens system. We use a VGG16 based U-net to capture both local and global information when predicting the resulting image. We create a dataset of training pairs of the same image with and without lens flare by adding synthetic lens flare to real HDR images. We developed our architecture to accept raw Bayer-array data as this data has been shown to capture more of the information native to the scene, and has a much higher dynamic range. The current trained model seems to eliminate rainbowing by adding to the two other channels (instead of subtracting from the over-represented channel). This unfortunately means that the PSNR of the synthetically corrupted signal is higher than that of the 'corrected signal' when compared to the ground truth. SSNR is also lower in the 'corrected signal'. This project is a work in progress and our future steps are discussed in the paper.

1. Introduction & Motivation

1.1. Lens Flare

Lens flare in photography occurs when rays from a very bright lights reflect and scatter around the camera lens system, resulting in starburst and rainbow patterns overlaid on the captured image. Illustrative examples of lens flares are included in figure 4. While lens flare is often added to photos for an artistic effect, or used to increase the perceived dynamic range of a scene, it can also detract from images by saturating or obscuring key subjects or creating unwanted artifacts.

Removing lens flare has commercial applications. Creating a lens stack or custom lenses is a very difficult endeavor and sometimes there are unexpected optical behaviors, such as excessive lens flares. In 2016, the google pixel phones were criticized by excessive lens flares, users were seeing a characteristic ring framing a lot of their images. Google added code to remove these artifacts bundled with the HDR+ mode [6]. It is unclear about how Google tackled this problem, but their description hints that the solution only concentrates on detecting and subtracting the rings around images.

At the time of capturing photos, lens flares can be avoided by not facing the camera at a shallow angle to the sun and by using a lens hood. A lens hood is similar to horse blinders for a camera; it will block a lot of the light that is not coming directly at the lens and will reduce stray light causing haze and internal reflections in the camera.

2. Related work

2.1. Adding Lens Flares to Images

Simulating the effect of lens flares in images is very popular as it increases the realism of rendered scenes and perceived dynamic range and brightness of captured images. Some creatives simply like lens flares; director J.J. Abrams gained a reputation for an extreme affinity to the effect. There also exist many online tools, such as befunky.com and photoshop tutorials that promise to add this effect to images. In order to add any lens flare to an image, you need to determine where the light sources are and their brightness. With the locations and brightness of the light sources, below are three different approaches for adding lens flare to the image.

2.1.1 Physics Based Rendering

With a high fidelity model of the camera lens system, one can run various physically-based ray-tracing algorithms from the inferred light location through the lens systems to generate the lens flare pattern that would be added to the
image. One recent paper in the subject of fast physics-based optics models, [5], show striking results for simulated lens flares, even accounting for the caustics of light falling through the lenses and much more. One of the major advantages of this approach is that, given such a model, one could simulate any camera system (existing or imaginary) to train the model on. One major disadvantage is that, in order for the simulations to behave similarly to the target camera system, the program must be given a relatively accurate model of the lens types, structures and materials. This information is all proprietary to the camera manufacturers and requires through investigation into dense IP documents, which might still yield incomplete information.

2.1.2 Capturing Real Lens Flare Images

A key insight in adding lens flare to images is realizing that the lens flare effect from light originating from a very bright sources is simply an additive effect to the scene (a pixel-wise addition). [8] trained a model to generate lens flares. Walch’s data collecting process involved using a bright projector to shine bright light at a camera from many different angles, thus capturing many lens flares. A similar approach, with some data augmentation, could also work or generating data for this project and is the logical next step.

2.1.3 Real time texture based rendering

Lens flares are a very popular effect in video games and photo editing software. Many simple real-time lens flare adders draw a vector from wherever the light source is to the center of the screen and add pre-computed textures to simulate the lens flare effect. Because of time constraints surrounding this project, I reverted from a physics based model to this simple texture addition model following roughly this OpenGL tutorial [7].

2.2. Learning Colors and Over-Saturated Pixels

The pixels affected by lens flares either become saturated or are simply the wrong color and brightness. [3] showed that they can use machine learning to fill in saturated pixels with believable patterns and increase the dynamic range of a single image. [3] decided to use a U-Net to combine information both from the tight area around the pixel to predict and the higher level features around it. We were motivated by their work to try this model. [3] decided where to inpaint based on the pixel brightness. Inpainting for, and correcting lens flares, is a harder problem because there could be pixels in perfectly valid colors and brightness that are still chromatically wrong. [1] used raw sensor data to be able to image in extreme low light conditions. Low light conditions degrade both color and brightness and so this research was a strong motivation to design a model that can train off of the raw pixel data.

3. Methods

3.1. Data

We are interested on training with raw images. There are several raw image datasets out there including RAISE [2] and the Google HDR+ dataset [4]. After looking at many
images from both, we decided that the HDR plus photos are much better quality and that high dynamic range images are appropriate for lens flares as they better mimic the situations in which lens flares would occur. To create our training pairs, we

1. download raw aligned frames from the HDR+ dataset
2. Filter for bright days based on upper bounding the film speed * ISO (as we are only simulating lens flares from bright days for now)
3. Convert the raw data into SRGB (a matrix operation)
4. Find the brightest spot in the image, and declare set it as the position of our fake sun, and render the lens flare
5. mosaic-ed the rendered lens flare grid (bring down from information about 3 channels to 1) and add it to the raw images.

3.2. Model

For this task, we use a U-net extremely similar to [3] with a VGG-16 encoder and deconvolutional decoder. A U-net structure was chosen for this per-pixel prediction task because both very local data, and regional as well as high level features, are relevant. The model is fully convolutional and so can accept any size inputs for training or predicting as long as they are a multiple of 32 in each direction. The skip connections concatenation along the channel axis, followed by a 1x1 px convolution to reduce the number of channels. Figure 2 shows the architecture of a very similar model. We trained the U-net using the reconstruction loss between the model output and the demosaiced image from the HDR+ dataset. Training with L2 loss resulted in washed out colors and images and so we switched to L1 loss.

4. Results

In this section we will analyze the weights learned by a network using L1 loss over 3 epochs. Figure 3 is over a test set example and shows (in clockwise order starting from top left) 1) a demosaiced version of the raw 'ground truth' data 2) a demosaiced version of the input to the model (ground truth + generated lens flare) 3) The absolute value of the difference between the data inputted into the model and the data outputted by the model 4) The output of the model. Looking at the difference image we notice that indeed in the output of the model the colors of the added lens flare are slightly muted. However, based on the representation of the difference between the images it appears that this is achieved, not by subtracting from the channel that is two high but rather by adding to the other two. It seems that the entire image is slightly brighter.

<table>
<thead>
<tr>
<th>Input</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>0.043570731 0.101278005</td>
</tr>
<tr>
<td>PSNR</td>
<td>13.31831654 17.03729157</td>
</tr>
<tr>
<td>SSIM</td>
<td>0.911874462 0.883254462</td>
</tr>
</tbody>
</table>

Table 1. The L1 (order 1 reconstruction loss), PSNR and SSIM comparing both the synthetically corrupted input and predicted output to the ground truth. Surprisingly the input outperforms on all. One hypothesis is that the network hasn’t quite figured out global tone mapping and will brighten most of the image, driving away the metrics from the ground truth.

5. Limitations

Currently, the lens flare model that we are using to generate training data is extremely limited. The same textures are rendered repeatedly yielding to a learning problem that is much too easy. This is perhaps a benefit when proving out the viability of the model, but replacing the training data with more realistic training pairs is the next top priority. Another limitation in the data is that the captured raw HDR data itself has lens flares. This data is supposed to be the clean, lens-flare-free ground truth for the model to train on. However, since this data is not perfect, the model will still learn to place lens flares in the final result. This is unavoidable unless one captures their own data with and without a lens hood or simulates data (which would bring it’s own lack of realism).

6. Conclusions and Future Work

In conclusion we were able to create a model which takes in raw image data and is able to visually temper the appearance of lens flares but was not successful in recreating the original lens-flare-free images. The next immediate steps are to keep the very simple lens flare model in order to prove out the viability of this model, more training and analysis with validation curves needs to be done. Currently, the model is very biased towards predicting an output very similar to the demosaiced input (given that the majority of the image area is unchanged by the lens flares patterns we added). The first technique to try would be to oversample areas with lens flares in the training examples. If the model is still not fully learning the structure of the lens flares, we can train a discriminator loss

$$\Delta_{\theta} = \frac{1}{m} \sum_{i=1}^{m} \log \left( D \left( G_{\theta}(x^{(i)}) \right) \right)$$

References

Figure 2. Figure from [3] describing their architecture. Our model is very similar.

Figure 3. Example output from algorithm. Images from left to right, up to down, are: 1) a demosaiced version of the raw ‘ground truth’ data 2) a demosaiced version of the input to the model (ground truth + generated lens flare) 3) The difference between the input to the model and the output to the model 4) The output of the model.

It seems that indeed in the output of the model the colors of the added lens flare are slightly muted. However, based on the representation of the difference between the images it appears that this is achieved, not by subtracting from the channel that is too high but rather by adding to the other two.


Figure 4. Example output from algorithm. Images from left to right, up to down, are: 1) a demosaiced version of the raw 'ground truth' data 2) a demosaiced version of the input to the model (ground truth + generated lens flare) 3) The difference between the input to the model and the output to the model 4) The output of the model.

It seems that indeed in the output of the model the colors of the added lens flare are slightly muted. However, based on the representation of the difference between the images it appears that this is achieved, not by subtracting from the channel that is too high but rather by adding to the other two.

A. External Image credits

A.1. Figure 4:

Left hand side:
flowers: https://torange.biz/it/lens-flare-holiday-lights-flower-tree-50338
meadow: https://upload.wikimedia.org/wikipedia/commons/3/3d/Lens_Flare.JPG
city at dusk: https://www.flickr.com/photos/137643065@N06/24244387051/

Right hand side:
Columns: https://9to5google.com/2016/10/25/google-acknowledges-lens-flare-issues-with-pixel/
Boats: https://www.androidcentral.com/google-pixel-lens-flare-what-it-and-how-avoid
Man: https://digital-photography-school.com/eliminating-lens-flare/

B. More Example Outputs

Figure 5. Example output from algorithm. Images from left to right, up to down, are: 1) a demosaiced version of the raw 'ground truth' data 2) a demosaiced version of the input to the model (ground truth + generated lens flare) 3) The difference between the input to the model and the output to the model 4) The output of the model.

Figure 6. Example output from algorithm. Images from left to right, up to down, are: 1) a demosaiced version of the raw 'ground truth' data 2) a demosaiced version of the input to the model (ground truth + generated lens flare) 3) The difference between the input to the model and the output to the model 4) The output of the model.