Automated Lens Flare Removal

Floris Chabert

Abstract

1 Motivation

Lens flare and ghosting can be prevalent artifacts when taking pictures of a scene with a direct bright light. Those artifacts are usually caused by internal reflections of the lens due to a thin anti reflective coating and can easily ruin a beautiful picture.

This project aims at automatically removing those lens artifacts via post-processing from a single input image to produce a restored picture. We designed an algorithm involving two steps: flare detection and recovery of the damaged region.

2 Related Work

Previous work found in the image processing literature around lens flare detection and generic flares detection can be separated in two categories: semi-automatic or using multiple images. Some flare detection algorithms involve either having a manual step where the user has to select the general area where the flare is present[1] or the specific color of the flare. This prevents false positive and makes the algorithm more robust. The second kind uses multiple pictures to detect flares. They use images with different exposures to be able to find the spots that saturated the sensor. Others use multiple frames with camera motion in between to figure out where the artifact is[2]. Another interesting category uses pictures with and without flash to detect general flares.

Various methods for recovery exist[3]. They are often referenced to as inpainting algorithms. Two major kind of inpainting are extensively documented: non-texture inpainting - often using partial differential equations based on different diffusion models[6][7] - and texture based methods[4]. Non texture based techniques usually work very well for small regions - especially when using higher degree derivatives to preserve edges - but have a tendency to produce blurred patches. Texture based inpainiting works better to fill larger holes as they copy-paste patches to recover the image by minimizing an error metric.

3 Method

Here we aim at an automated detection of the flares using a single input image. This involves a custom blob detection algorithm based on a concept used in OpenCV[5] tuned for the specific lens flares we want to detect and a hybrid inpainting method called exemplare-based inpainting[8].

3.1 Detection

The chosen detection algorithm used includes five main steps:

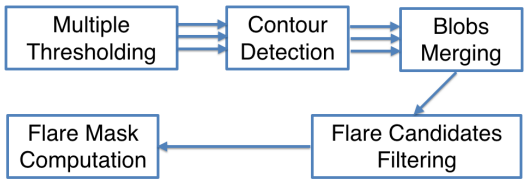

Figure 1: Flare detection algorithm

Multiple Thresholding The image is converted to grayscale and binarized using a range of thresholds.

Contour Detection For each binary image, we then find the contours using a border following method[9].

Blob Merging The center of each blob is then computed and blobs from the different binary images are merged depending on their distance and similarity. We finally obtain a set of potential flare candidates.

Flare Candidates Filtering The flare candidates are pruned using various metrics which parameters have been tuned using a set of images as to be robust while avoiding false positive. Those metrics include circularity of the blob, convexity, inertia and area.

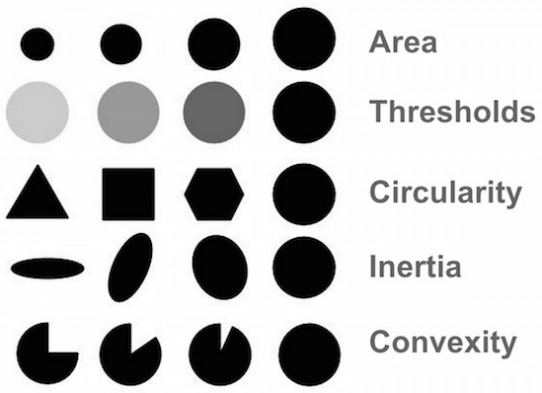

Figure 2: Impact of filtering parameters for blob detection[5]

Flare Mask Computation Finally the mask selecting the flares is computed for the next step.

3.2 Recovery

After the flare mask has been computed we can recover the damaged area using exemplar-based inpainting.

After selecting a window around the flare - to avoid searching over the whole image, assuming good texture candidates are near

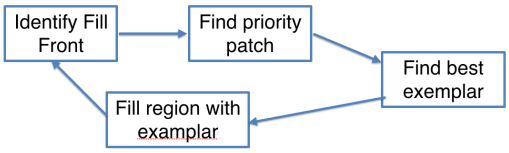

Figure 3: Flare inpainting algorithm

the missing pixels - we execute the following algorithm until all the pixels have been recovered:

Identify Fill Front We first find the contour of the region we want to fill.

Identify Priority Patches Patches on the fill front are assigned priorities as to privilege patches that continue strong edges and are surrounded by high confidence pixels.

Find Best Exemplar By priority order, we then search the window for known patches that minimize the error.

Fill Region using Exemplar Patch We finally select pixels from the best patch to fill the masked pixels in the current patch to recover.

- Extract the manually selected initial front $\delta\Omega^0$.
- Repeat until done:

1a. Identify the fill front $\delta\Omega^t$. If $\Omega^t = \emptyset$, exit.
1b. Compute priorities $P(\mathbf{p}) \quad \forall \mathbf{p} \in \delta\Omega^t$.
2a. Find the patch $\Psi_{\hat{\mathbf{p}}}$ with the maximum priority, *i.e.*, $\hat{\mathbf{p}} = \arg\max_{\mathbf{p} \in \delta\Omega^t} P(\mathbf{p})$.
2b. Find the exemplar $\Psi_{\hat{\mathbf{q}}} \in \Phi$ that minimizes $d(\Psi_{\hat{\mathbf{p}}}, \Psi_{\hat{\mathbf{q}}})$.
2c. Copy image data from $\Psi_{\hat{\mathbf{q}}}$ to $\Psi_{\hat{\mathbf{p}}}$ $\forall \mathbf{p} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$.
3. Update $C(\mathbf{p}) \quad \forall \mathbf{p} \in \Psi_{\hat{\mathbf{p}}} \cap \Omega$

Figure 4: Exemplar-based inpainting steps[8]

4 Results

The described method show good results. The detection is robust and finds the flares is the vast majority of picture from the test-

ing set. The exemplare-based technique for recovery is also very solid and allows filling of larger regions without blurring the image.

Figure 5: Image 1 before processing

Figure 6: Image 1 after processing

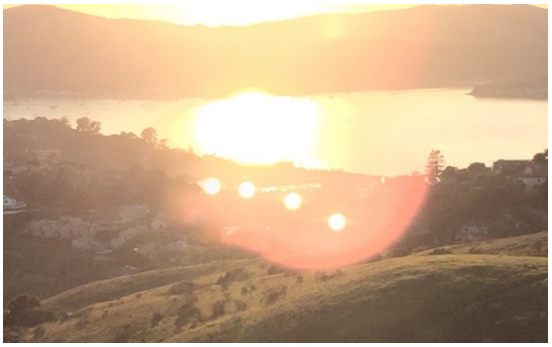

Figure 7: Image 2 before processing

The full algorithm has been implemented in C++ and runs in less than 5 seconds on a Intel-core i5 processor. It has also been ported to an arm64 platform and runs in less than 10 seconds on an iPhone 6s.

Figure 8: Image 2 after processing

5 Discussion

Different detection algorithm have been tried for this project. Circular detection through Hough-transform has not been very robust as most flares are not fully circular. A SIFT descriptor based method has also been tested. A set of flare descriptors were learned from a training set and then matched against keypoints in the target image. Unfortunately this method was too dependent on the training set.

The chosen blob detection method ends up being quite robust. Some issue exist though. The main one being false positives in specific scene. The following figure shows a false detection were a green traffic light in the fog is wrongly thought to be a flare.

Restoration has also been tried using non texture inpainting. Unfortunately the large size of the flares makes the recovered area quite blurry. Here you can see the difference between a non-texture recovery (using Navier-stokes) compared to the exemplare-based method.

Future work on this subject include better detection methods to prevent the false positive issue described above as well as bet-

Figure 9: Traffic light wrongly detected as a flare

Figure 10: Recovery using non-texture based inpainting

ter inpainting using modern techniques. Another area of research would look at using this method for other kind of flares as well as using the mask of the detected flares to find nearby regions containing flares.

References

[1] usan Psotny, *Removing lens flare from digital photographs*, Charles University in Prague, Diploma Thesis

[2] ndreas Nussberger, Helmut Grabner, Luc Van Gool, *Robust Aerial Object Tracking in Images with Lens Flare*, Comput. Vision Lab., ETH Zurich

Figure 11: Recovery using exemplare based inpainting

[3] arcelo Bertalmo, Vicent Caselles, Simon Masnou, Guillermo Sapiro, *Inpainting*

[4] ajul Suthar, Mr. Krunal R. Patel, *A Survey on Various Image Inpainting Techniques to Restore Image*, Int. Journal of Engineering Research and Applications

[5] atya Mallick *Blob Detection Using OpenCV*, learnopencv.com

[6] ony F. Chan, Jianhong Shen *Mathematical Models for Local Nontexture Inpaintings* SIAM Journal on Applied Mathematics, Vol. 62, No. 3

[7] ony F. Chan, Jianhong Shen, *Non-Texture Inpainting by Curvature-Driven-Diffusions*, Visual Comm Image Rep 06/2001

[8] iansheng Liu, Mingming Li, Fangfang He *Region Filling and Object Removal by Exemplar-Based Image Inpainting*, IEEE Transactions on Image Processing, Vol. 13, No. 9, 2004

[9] uzuki, S. and Abe, K., *Topological Structural Analysis of Digitized Binary Images by Border Following* CVGIP 30 1, pp 32-46, 1985