Real Time Video Haze Removal Using Dark Channel Prior

Group 9: RenjieLuo TianLuo BoXue BowenMu

***Executive Overview*--In our final project, we implemented a method of real time video haze removal by using dark channel prior[1].**

Haze is a critical problem for computer vision, especially haze weather happened much more frequently right now. The image with haze would cause the critical content missing in the image and influence the target recognition. Under terrible haze weather, the image contrast would be lower, the content would be fuzzy and the image color would be distorted. Thus haze removal would be useful to cut costs and increase the accuracy of the recognition. We use the dark channel prior as the basic method to erase the haze, especially the uneven haze. For the most non-sky area, there is at least one color channel that has very low intensity which close to zero. For the image with haze, The dark channel will have higher intensity in regions with denser haze. The intensity of dark channel is rough approximation of the thickness of the haze. By using this theory, we first find the minimum value with a specific window size in each channel of the image and pick the smallest value for each channel to form the dark channel model. In this way we can restore the image without haze. To increase the speed, we used the Streaming Maximum-Minimum Filter[2], which uses only less than three comparisons per element to calculate the minimum and maximum value in an array. In order to reduce the halo artifacts, we implemented Guided Filter[3], a bilateral filter that preserves edges in the smoothing process, which is much faster than the soft matting. As this is not fast enough to achieve real time video dehazing, we used CUDA for GPU acceleration, which is much faster than python implementation.

Besides the algorithm implementation, we also evaluated the results by applying this method to plenty of application scenarios and found that our method works well enough in most light haze circumstances. In thick haze images, the missing information is too much to restore images without haze.

I. Background and Impact

With the development of the industrialization, the Haze has happened more frequently in recent years due to the adverse influence on the atmosphere environment. Under the Haze weather, the adjusting contrast of the image would be lower, the content would be fuzzy, and the image color would be distorted. For the general public, Haze could influence the photography that the backgrounds of images and videos would be fuzzy. For the industry, Haze could cause the critical content missing in the image and influence the application of computer vision. For target recognition, the edge of the objects could not be found correctly from the image in the haze weather which would be critical for technology like automatic vehicle. The camera of the vehicle was always used for detecting the distance between the object and avoiding the collision with others. Under the haze weather, the camera could make the inaccurate decision and cause serious consequences.

Therefore, it is necessary to implement a way to remove the haze in the real-time to assist the application of computer vision.

Since it is almost impossible to eliminate the influence of the mist weather from outside, the common method is applying multiple algorithms and filters the restore the original image from the video. However, in the real world, the Haze was normally formed unevenly which is very hard to erase by applying a single filter. The team proposes a new idea and application for Haze removal that would reduce the cost and increase the accuracy of the object recognition.

II. Method

A. Dark prior

In most of the non-sky patches, there are some pixels having extreme low intensity in at least one color channel. Which indicates that the minimum value of light intensity in this patch is a small value.[1] The original image with haze and dark channel image are shown below.

Figure 1 the image with Haze

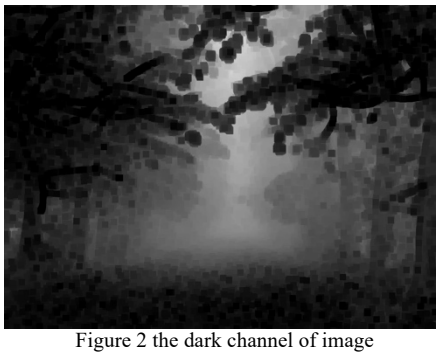

Figure 2 the dark channel of image

We could get a mathematical definition for dark channel, for an input image J, its dark channel could be expressed as:

$$\mathrm{J}^{\mathrm{dark}}(x) = \min_{y \in \Omega(x)} \left(\min_{c \in (r,g,b)} J^{c}(y) \right) \qquad (1)$$

Where J^C is a color channel of J and Ω(x) is a local patch centered at x.

Dark channel prior theory indicates that the intensity of J^{dark} is low and tend to be zero.

$$J^{dark} \to 0 \qquad (2)$$

In real life, there are three main factors that cause low channel intensity in dark priors: (1) shadows (2)colorful objects or surfaces (3) dark objects or surfaces

In short, natural out-door images are full of shadows and colorful. Therefore, the dark channels of these images are always very dark.

We have tried several images and we could clearly see that dark channel prior is general.

In the paper, the author had analyzed character of more than 5000 images and almost all of them are consistent with this prior. Therefore, we could regard it as a theorem.

With this dark channel prior, we need derive some mathematical steps to solve this problem.

At first, in computer vision, there is a widely used model to describe the formation of a haze image[4]:

$$I(x) = J(x)t(x) + A\big(1 - t(x)\big) \qquad (3)$$

Where I(x) denotes the original image (the image to be de-haze), J(x) is the haze-free image we want to recover, A is the global atmospheric light, and t(x) is the medium transmission describing the portion of the light that is not scattered and reaches the camera. Obviously, this is an equation with infinite solutions. This is why dark channel prior is needed.

Equation (3) is equivalent to

$$\frac{I^c(x)}{A^c} = t(x)\frac{J^c(x)}{A^c} + 1 - t(x) \qquad (4)$$

Assume global atmospheric light A is given and medium transmission t(x) is constant for local patches which is denoted as $\tilde{t}(x)$.

Taking the minimum operation twice for both side of equation (4), we would have the following equation:

$$\min_{y\in\Omega(x)}\left(\min_c \frac{I^c(y)}{A^c}\right) = \tilde{t}(x)\min_{y\in\Omega(x)}\left(\min_c \frac{J^c(y)}{A^c}\right) + 1 - \tilde{t}(x) \quad (5)$$

In the above equation, J is the haze-free image we want to recover. According to dark channel prior theory as we discussed before, we would have:

$$J^{dark}(x) = \min_{y\in\Omega(x)}\left(\min_{c\in(r,g,b)} J^c(y)\right) = 0 \qquad (6)$$

This leads to (10):

$$\min_{y\in\Omega(x)}\left(\min_c \frac{J^c(x)}{A^c}\right) = 0 \qquad (7)$$

Putting Equation (7) into Equation (5), we would have: (8)

$$\tilde{t}(x) = 1 - \min_{y\in\Omega(x)}\left(\min_c \frac{J^c(y)}{A^c}\right) \qquad (8)$$

which is the estimation of transmission $\tilde{t}(x)$.

In practice, there always exist some particles in the atmospheres even in clear days. Therefore, the objects in the distance can still be influenced by haze. In addition, the existence of haze makes human to perceive depth, which is called aerial perspective. So it is necessary to keep a vary small amount of haze for the distant objects. This can be done by introducing a constant parameter ω into equation(8):

$$\tilde{t}(x) = 1 - \omega \min_{y\in\Omega(x)}\left(\min_c \frac{J^c(y)}{A^c}\right) \qquad (9)$$

In the above inferences, we assumed that the global atmospheric light A is given. In practice, we could obtain this value by applying dark channel prior to haze-image. Specific steps are as follows:

Pick the first 0.1% of the pixels from the dark channel according to intensity.

Among these pixels, the highest intensity of the corresponding pixel in original image I(x) is selected as atmospheric light A.

At this point, we could recover the haze-free image. From equation (1), we know that: J = (I-A)/t + A.

We have already get values of I, A and t, which indicates the calculation of J could be done completely.

When the transmission t(x) is close to zero, the scene radiance J would be very large. Resulting in recovered scene radiance J prone to noise. Therefore, we could set a low threshold t_0 for transmission t(x). The final scene radiance J(x) is as follows:

$$J(x) = \frac{I(x) - A}{\max(t(x), t_0)} + A \qquad (10)$$

Haze removed image is shown below:

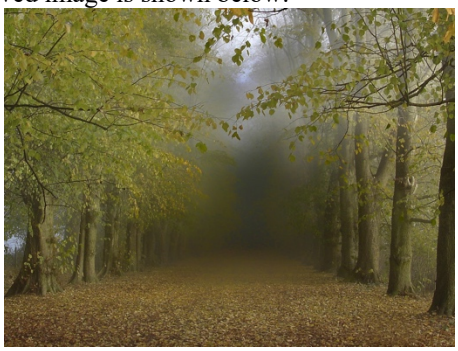

Figure 3 the image with haze removal

B. Streaming Maximum-Minimum Filter

Streaming Maximum-Minimum Filter: It could use no more than Three Comparisons per element to calculate the minimum and maximum value in an array. For this algorithm, the team defined the steam latency of as filter as the maximum number of the data points required after a specific window passed[2]. The stream latency is a method that could calculate the highest frequency of the data appears. For the application of Haze removal, Team assumes the size of the window would be strictly less than the number of pixels in each row.

The algorithm is that team created 2 queues which have 1 in each queue to represent the max and min value. Then, for each pixel in an array, if the position of the pixel is larger than the window size, we output the front value of each queue as the

maximum and minimum range. If the current pixel is large than the pixel on its left, and while the current pixel is larger than one of the values in the queue, these passed value would be removed from the queue. The same method would be applied from the front of the queue. Then the max and min value would be the last value from the queue.

This method has a O(1) speed level and increase the efficient of calculation when applied on algorithm for dark channel image.

C. Guided Filter

Edge-preserving filter is an important for computer vision. It would denoising, smoothing and sharpening the image. The guided filter assume there is a linear model between the guidance I and the filtering output q. Team assume that q is a linear transform of I in a window w_k centered at the pixel k:

$$q_i = a_k I_i + b_k, i \in w_k \quad (11)$$

Where a and b are the linear coefficients assumed to be constant in wk. In this method, team use the solution of the minimizes the difference between q and p while maintain the linear model. Then, the solution would be:

$$a_k = \frac{\mathrm{cov}(I,p)}{\mathrm{var}(I)+\epsilon} \quad (12)$$

$$b_k = \overline{p_k} - a_k \mu_k \quad (13)$$

$$q_i = \frac{1}{|\omega|}\sum_{k|i\in\omega_k}(a_k I_i + b_k) \quad (14)$$

$$q_i = \overline{a_i} I_i + \overline{b_i} \quad (15)$$

Where a_i and b_i are the average coefficients of all windows overlapping i.

III. Prototype

A. Python Implementation

Team first decided to implement the haze removal method in python environment. Team first using the Dark Channel Prior method to generate a dark channel image.

Since The image with haze was formed by the original image was formed by the equation:

$$I(x) = J(x)t(x) + A\big(1 - t(x)\big) \quad (3)$$

I(x) is the image with Haze, x is a coordinate for a point, t(x)is the transmission rate, A is the atmospheric lightness. and J(x) is the image without Haze. By calculating the following values, we could restore the image without haze. When calculating the transmission rate, team need to apply a soft matting method to erase the halo artifacts around the edge. To increase the speed and seek a better result, the team decided to apply a guided filter which is a bilateral filter that could preserve edges in the smoothing process.

After applying the following steps, we program the prototype on a GPU to achieve the real-time Haze remove.

B. C ++ and CUDA Implementation

Even though python has gained a lot of popularity in modern computer vision field, however, due to its scripting property, it's too slow to meet the need of real time performance. For faster performance of the algorithm, we used GPU combined with OpenCV [5] CUDA library. And since OpenCV CUDA library [6] is only available via C ++, we code this using Visual Studio C++.

The coding process is almost the same as the python implementation. But there are a few details we need pay attention to, unless the CUDA[7] implementation may not be beneficial to improve the performance of the algorithm.

Data structure: The data structure on GPU VRAM is completely different from CPU RAM. Usually, the sequential instruction is not preferred on GPU and the common sequential loop is rarely used in GPU implementation. As a result of this, we used GpuMat in CUDA library which is the equivalent form of Mat in ordinary OpenCV library. A nuance is that GpuMat cannot be indexed through entries, and we should always consider it as a vector or matrix to be operated on.

Uploading time between GPU and CPU: the original data we use should be first transfer from RAM to VRAM. But because of the bandwidth limitation between them, the time we need to transfer data is relatively high compared to the computational time on GPU. Therefore, we should transfer the data between CPU and GPU as few times as possible. In our implementation the data has been transferred between them only twice.

Parallel computation: The core idea that makes the GPU perform faster than CPU is that it makes use of the idea of parallel computation, otherwise GPU may not be a better choice in terms of the time cost. In our implementation, the guided filter, the streaming Maximum-minimum filter and even the dark prior computation can benefit from the advantage of parallel computation. But for computing the atmospheric air light, only sequential implementation is available, so we handle this via CPU.

CUDA initialization: Every time when we start a CUDA program, it takes some time for CPU to send signal to GPU so that the CUDA environment can be initialized. In our experiment, we found that it usually took 1.5-2s for the initialization process to finish. Thus, if we only need process only one image, A pure CPU implementation is definitely faster than GPU implementation. The advantage of CUDA can be seen only when we capture real time video or process a huge amount of images or large video sets. In our implementation, we only use CUDA program to process data frame from a facetime camera. After all, the video we output can reach the speed of nearly 20 frames per second.

IV. Result

Our experiment evaluation results are shown below. The upper images are the original image with haze, and the lower images are images after haze removal. The former two sets are real time video implementation, while the remaining two sets are video clips from the internet. The first set is real time video with low

intensity haze, while the second set is with high intensity haze. The third set is a video with less sky area and relatively light haze, while the last set captures the scenario in which haze is thick and sky area is large.

Figure 4 Real time video with low density haze

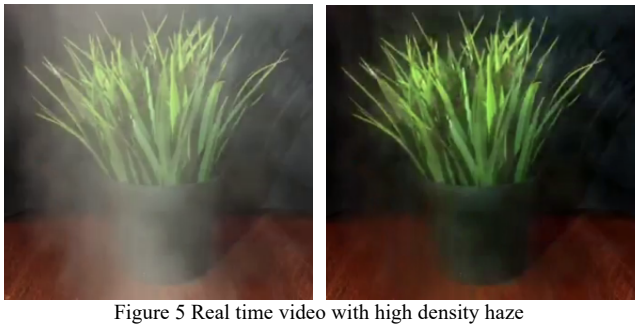

Figure 5 Real time video with high density haze

Figure 6 Video from the Internet with light haze and less sky area

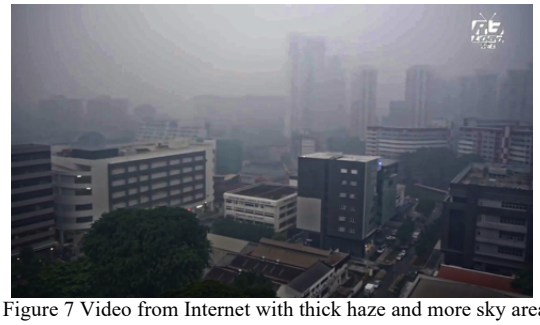

Figure 7 Video from Internet with thick haze and more sky area

From the above results, we can see that in real time video haze removal, our algorithm works well in light haze circumstances but the results under thick haze are still acceptable, of which circumstance the haze cover lots of information so that we cannot produce the haze removed image without much information loss. In the third set of results, we can see some halo surrounding the fan, which is generated because there is not a set of parameters in Guided Filter that suits all the frame. In fourth result set, we can see our method suits well for non-sky areas (the buildings in the image) but not well for sky area (the buildings that are far away) because the dark channel for sky areas (or very "white" areas) is almost white everywhere, which is regarded as a whole haze by our method.

V. References

[1] He, Kaiming, Jian Sun, and Xiaoou Tang. "Single image haze removal using dark channel prior." *IEEE transactions on pattern analysis and machine intelligence* 33.12 (2011): 2341-2353

[2] Lemire, Daniel. "Streaming maximum-minimum filter using no more than three comparisons per element." *arXiv preprint cs/0610046* (2006).

[3] He, Kaiming, Jian Sun, and Xiaoou Tang. "Guided image filtering." *European conference on computer vision*. Springer, Berlin, Heidelberg, 2010.

[4] Narasimhan, Srinivasa. *Models and Algorithms for Vision Through the Atmospere*. Columbia University, 2004.

[5] *OpenCV Library https://docs.opencv.org/3.4/index.html*

[6] *OpenCV CUDA Library* https://docs.opencv.org/3.4/d1/d1e/

[7] *NIVIDIA CUDA developer Guide.*