PA1 – Multi Image Denoising

20231126 황산하

1. Introduction

본 과제는 연속으로 한 방향으로 변하며 찍힌 여러 개의 이미지를 이용해서 노이즈를 제거하고 이미지의 품질을 높이는 Image Denoising을 실습하는 과제이다. 먼저 이미지를 비교하며 Disparity를 구하고, 이 Disparity를 이용해 Cost Volume을 만든다. 그리고 픽셀 단위로 Cost volume을 조회하여 가장 Disparity가 작은, 즉 품질이 괜찮은 Disparity를 찾을 수 있다. 여기서 구한 Disparity를 통해 각각의 이미지들을 warping 시키고 최종적으로 한번 더 집계하는 과정을 통해 Denoising을 달성할 수 있다.

정리하자면 1) Disparity를 구하고, 2) 이를 depth만큼 쌓아 Cost Volume만들어 3) Semi global matching 알고리즘을 통해 가장 품질이 괜찮은 disparity를 찾아 4) 이미지 와핑으로 좌표를 이동시키는 작업을 진행했다.

2. Code

제출한 코드는 크게 5가지의 스크립트로 구성되어있다.

1) **test.py**

이 코드는 작성한 코드가 잘 동작하는지 간단하게 돌려본 실험용 코드 스크립트이다. 전체 이미지에 대해 돌려보려면 시간이 너무 오래 걸리기 때문에 이미지 두개로만 실험해 보기 위해 작성한 코드이다. 즉 실제 알고리즘 플로우에는 포함되지 않는다.

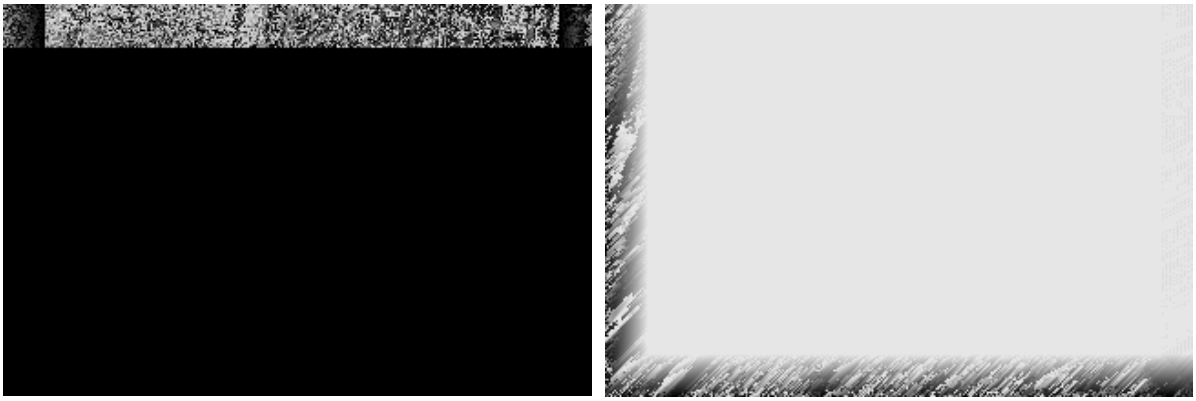
2) **similarity.py :** Dispartiy와 Cost volume 구하기

이 스크립트는 Disparity를 구할 수 있는 각각 다른 방법들을 구현해 놓았다. SAD(Sum of Absolute Differences)와 SSD (Sum of Squared Differences)를 이용했다. 이 두가지 방법은 말 그대로 이미지 간의 차이의 절대값, 차이의 제곱을 이용해서 구한다.

3) **aggregate_cost_volume.py** : Dynamic Programming을 통해 Semi global matching 알고리즘 구현

이 스크립트는 Semi global matching이 동작하는 코드 스크립트이다. 재귀함수로 코드를 작성하면 컴퓨팅 파워가 굉장히 많이 들고 비효율적이기 때문에 DP의 Memoization방법을 이용해 시간을 대폭 줄일 수 있었다. 이 때 Memoization을 위해 전역 변수로 MEMO라는 Cost Volume과 같은 사이즈의 행렬을 INF값으로 채워진 상태로 생성해서 이용했다. (여기서는 적당히 크도록 9999999로 선언해주었다) 또한 최솟값을 찾기 위한 변형된 quick sort를 이용해 속도를 좀 더 높여보려고 했고, 이 외에도 Semi global matching에 필요한 여러 custom 함수들이 작성 되어있다.

Semi global matching을 진행할 때, PPT에는 Depth 방향으로 조회를 하도록 설명이 되어있었다. 처음 알고리즘을 구현할 때, 방향을 먼저 결정하고, Depth를 하나씩 조회해서 x와 y로 먼저 보도록 알고리즘을 만들었다. 왜냐하면 결국에는 전체를 전부 조회하고, 8방향을 합치는 알고리즘이다 보니 순서는 관계없다고 생각했기 때문이다. 하지만 Disparity를 시각화 해보니 생각했던 것과는 많이 달랐다. 이미지는 좌로 움직이고 있는데 이를 위의 방향으로 조회하니 좀 상상과는 다른 결과를 확인할 수 있었다. 결국에는 PPT에 나와있는 방향대로 구현을 완료했다. Python의 for문으로 구현을 해서 그런지 생각보다 속도가 많이 느렸다. 최적화를 위해 미리 좌표에 대한 정보가 들어있는 Window를 (depth, height, width)의 형식으로 리스트에 저장을 하고, 이를 불러와 현재 state를 결정하여 Semi global matching을 진행했다. 방향은 forward pass에서는 좌, 좌위, 위, 우위 backward pass에서 우, 우아래, 아래, 좌아래를 조회하도록 만들었다. DP를 진행하다보면 indexing error가 발생할 수 있다. Depth와 y,x좌표에서 주변값을 탐색하다보니 Memo에 없는 값을 조회하려고 할 때가 있는데, 이는 INF값을 받아오도록 check_memo라는 함수로 예외처리 해주었다.



[그림 1] 잘못 구한 Disparity 예시

[그림 2] 제대로 구한 Disparity

4) **warp.py** : 이미지 와핑

이 스크립트는 이미지 와핑을 하는 코드이다. 이미지의 좌표를 조회해서 disparity만큼 이동시켜주었다. 이동시킬때 원본이미지에 없는 부분이면 조회가 불가능하기 때문에 예외처리가 포함되어있다. 이미지 와핑은 각각의 이미지와 레퍼런스 이미지 사이의 aggregated disparity를 이용하여 이미지 안의 값을 이동시켜주는 방식으로 동작한다. 레퍼런스 이미지가 가운데 4번 이미지 이므로 1~3번 이미지는 값을 더해주고, 5~7번 이미지는 값을 빼주어 이동시킨다.

5) **Semi_Global_Matching.py**

이 스크립트는 이번 PA의 Main 코드라고 볼 수 있다. 지금껏 작성했던 코드들이 여기에서 호출이 되어 Image Denoising을 진행하고 결과를 확인할 수 있는 코드이다. 레퍼런스 이미지를 제외하고 6장을 처리하는데 걸리는 시간은 약 15분정도 걸리도록 구현했다. (한 이미지 처리하는데 2분 30초 정도)

3. Result

Ground Truth 이미지와의 loss 계산을 통해 내가 만든 알고리즘의 성능을 확인할 수 있었다. 처음에는 아쉽게도 MSE는 1065.66, PSNR은 17.85가 나왔다. 그냥 원본이미지에 아무 알고리즘을 적용하지 않고 합쳐도 MSE는 322, PSNR은 22가 나오는데 어딘가 잘못되었을 거라고 생각하고 알고리즘을 다시 생각했다. 아래의 그림 2 (좌)에서 볼 수 있듯이 오히려 노이즈가 심해진 것을 알 수 있sum of squared differences다. 오른쪽은 아무 알고리즘을 거치지 않고 aggregation을 적용했을 때 얻을 수 있는 사진이다.

[그림 3] 처음 생각했던 알고리즘 (좌), 어떤 알고리즘도 거치지 않고 합쳤을 때 (우)

이 상황을 개선하기 위해 처음으로 돌아가서 disparity를 구할 때 실수하지는 않았는지, Semi global matching이 내가 생각한대로 동작하고 있는지 확인했다.

이에 따라 몇가지 비효율적으로 메모리를 사용하고 있는 부분과 잘못된 부분을 발견할 수 있었는데, 내가 생각했던 것과는 달리 와핑을 반대 방향으로 해주고 있었고, 앞에서 말했듯이 Semi global matching을 할 때 조회하는 방식 및 방향에 혼동이 와서 정확하게 값을 구할 수가 없었다. 그래서 최소한 PPT와 동일하게 작동하도록 미리 좌표를 생성하는 generate_cor_pass()라는 함수를 이용해 좌표들을 생성했으며 이를 하나씩 조회해서 forward pass와 backward pass를 진행했다. 좌표를 생성한 순서를 통해 진행되는 방향을 결정할 수 있다는 것을 깨닫고는 PPT와 같은 방식으로 방향이 진행되도록 코드를 고쳤다. 그래서 결국 [그림 2]에서 얻은 disparity를 구할 수 있었고 이에 따라 최종 결과물을 [그림 4]와 같이 얻을 수 있었다. [그림 4]에서 얻은 결과물은 PSNR 24.82, MSE 214로 지금껏 얻은 결과물 중에 가장 좋은 결과물이다.

4. Conclusion

이번 과제를 진행하면서 가장 어려웠던 점은 이미지 행렬을 다루는 것이었다. 나에게 익숙한 방식으로 코드를 만들다 보니까 주어진 코드와는 다르게 Depth가 가장 앞으로 나와 (D,H,W) 이런 모양의 Cost volume을 얻을 수 있었고, 시작을 이렇게 하다 보니 뒤에 나오는 Semi global matching 알고리즘의 방향을 다루는 부분이라든지, 커널을 이용해서 조회를 해야하는데 방향이 달라져서 애를 먹는다든지 하는 문제가 계속해서 발생했다. 또한 Dynamic Programming에서 Index error가 왜 발생하는지 처음에 알아채지 못해서 꼬박 며칠을 고민했었다. 천천히 고민해보니 당연히 그렇게 나올 수 밖에 없는 구조였는데 말이다.

이런 여러 문제를 해결하기 위해 디버깅을 통해 달라지는 cost volume과 disparity map을 보면서 어디가 에러가 나는지, 생각한대로 동작하고 있는지 확인하며 최대한 생각했던 방법으로 구현 하려고 했다. 문제를 해결하는 과정에서 이미지 데이터에 대한 이해가 늘었고, 조금 더 속도를 높이려면 어떤 방식으로 할 수 있을지를 고민하다 보니 코딩에 대한 자신감도 늘었다.

기존에 이미지를 다루는 방법은 레퍼런스에 의존해서 pytorch 딥러닝 모델을 통해 어떻게든 결과만 받아보려고 했다면, 지금은 이미지의 데이터가 왜 3차원으로 생성됐고 이것이 의미하는 것이 무엇인지 이해하는 과정 덕분에 딥러닝이 아니어도 이미지 데이터를 이렇게까지 다룰 수 있구나 느낄 수 있었던 과제였다.

[그림 4] 최종 이미지