**Computer Vision HW4 Report**

Student ID: R09942174

Name: 黃繼綸

**Visualize the disparity map of 4 testing images.**

|  |  |
| --- | --- |
| Tsukuba | Venus |
|  |  |
| Teddy | Cones |
|  |  |

**Report the bad pixel ratio of 2 testing images with given ground truth (Tsukuba/Teddy).**

|  |  |
| --- | --- |
|  | bad pixel ratio |
| Tsukuba | 4.04% |
| Teddy | 9.61% |

**Describe your algorithm in terms of 4-step pipeline.**

1. **Cost computation**

First, I used the brute force method to implement the census cost estimation. However, the implementation time was over time (around 17 mins). So, I tried to improve this algorithm. I first computed the local binary matrix from both the left and right images. By doing so, I could use the matrix operation to accelerate the implementation time (around 25 s).

1. **Cost aggregation**

I used the bilateral filters to filter out the disparity maps. The parameters of the jointBilateralFilter is important. I have tried several parameters, as shown below:

|  |  |
| --- | --- |
| Tsukuba image | Bad pixel ratio |
| jointBilateralFilter(30, 5, 5) | 4.52% |
| jointBilateralFilter(9, 5, 5) | 4.36% |
| jointBilateralFilter(30, 10, 10) | 4.81% |
| jointBilateralFilter(20, 5, 5) | 4.27% |
| jointBilateralFilter(20, 10, 10) | **4.04%** |

1. **Disparity optimization**

I used the winner\_take\_all algorithm, which can be implemented by argmin.

    winner\_L = np.argmin(cost\_list\_L, axis=2)

    winner\_R = np.argmin(cost\_list\_R, axis=2)

1. **Disparity refinement**

First, I used the left-right consistency check method to enhance the quality of disparity map, as shown below.

for i in range(h):

        for j in range(w):

            if winner\_L[i, j] == winner\_R[i, j - winner\_L[i, j]]:

                continue

            else:

                winner\_L[i, j]=-1

Second, I used hole filling to fill out the invalid disparity map generated from the left-right consistency check method, as shown below.

for i in range(h):

        for j in range(w):

            if winner\_L[i, j] == -1:

                l\_idx = j - 1

                r\_idx = j + 1

                while l\_idx >= 0 and winner\_L[i, l\_idx] == -1:

                    l\_idx -= 1

                if l\_idx < 0:

                    FL = 100000000

                else:

                    FL = winner\_L[i, l\_idx]

                while r\_idx < w and winner\_L[i, r\_idx] == -1:

                    r\_idx += 1

                if r\_idx > w - 1:

                    FR = 100000000

                else:

                    FR = winner\_L[i, r\_idx]

                winner\_L[i, j] = min(FL, FR)

Finally, the weightedMedianFilter is used to enhance the disparity map.

labels = xip.weightedMedianFilter(Il.astype(np.uint8), winner\_L.astype(np.uint8), 18, 1)