3D DATA PROCESSING - Assignment 1

Giuseppe Labate
mat.2096556

1. Introduction

In the assignment 1 it was requested to generate the disparity map ($\boldsymbol{d_{sgm}}$) for a set of given images by implementing the Semi-Global Matching (SGM) stereo matching algorithm.
It was provided the initial guess of the disparity map (***mono_***) defined up to a scale factor and calculated with data-driven monocular depth estimation method.
The task involved determining the scalar factor ***X*** by using the disparities calculated with the SGM that exhibited good confidence.
Finally, the disparity map was refined using the initial guess, now enhanced by the accurately determined scalar factor ***X***.

2. Implementation

Task 1

For this task it has been implemented the cost function formula:

$$E(p,d) = E_{data}(p,d) + E_{smooth}(p,q) - \min_{0 \le \Delta \le d_{max}} E(q,\Delta)$$

- To calculate $\boldsymbol{E_{data}(p,d)}$ the cost_ vector was picked to access to the cost volume.
- $\boldsymbol{E_{smooth}(p,q)}$ it's the minimum of three formulas that are quite easily implemented by simply following their definitions:

$$E_{smooth}(p,q) = min \begin{cases} E(q,f_q) & if\ f_q = f_p \\ E(q,f_q) + c_1 & if\ |f_q - f_p| = 1 \\ \min\limits_{0 \le \Delta \le d_{max}} E(q,\Delta) + c_2 & if\ |f_q - f_p| > 1 \end{cases}$$

- $\boldsymbol{\min_{0 \le \Delta \le d_{max}} E(q,d)}$ is calculated by finding the minimum cost value for the previous pixel in the path along all disparities.

To find the previous pixel in any current path, one can simply subtract, from the current coordinates, the directions belonging to the current path
So $path\_cost\_[cur\_path][cur\_y - dy][cur\_x - dx][d]$, where $dy\ and\ dx$ are the *paths_[curr_path].direction_y* and *paths_[curr_path].direction_x.*

The first pixel in a path is a special case of the *compute_path_cost* function, so it must be managed properly.
For this pixel, it can't be accessed the previous one (q), so the path cost is reduced to $E(p,d) = E_{data}(p,d)$.

```
// if the processed pixel is the first:
if(cur_y == pw_.north || cur_y == pw_.south || cur_x == pw_.east || cur_x == pw_.west)
{
  //Don't consider smoothness term
  for(int d = 0; d < disparity_range_; d++)
  {
    path_cost_[cur_path][cur_y][cur_x][d] = cost_[cur_y][cur_x][d];  //consider only volume cost  E(pi,d) = E data(pi,d)
  }
}
```

Figure 1: If condition that manages the first pixel in the path case.

Task 2

In this task it was asked to correctly initialize the variables to call compute_path_cost for each pixel coordinates and path.

First idea

Here is where my coding process got some troubles.
In fact, at first, my idea was to scan every pixel in the image. To do so, I've initialized the variables as in the *Figure 2* below:

```
//initialize the variables for x depending on the case
if(dir_x == 1 || dir_x == 0)
{
  start_x = 0;
  end_x = width_;
  step_x = 1;
}
else //(dir_x == -1)
{
  start_x = width_ - 1;
  end_x = -1;
  step_x = -1;
}
```

```
//initialize the variables for y depending on the case
if(dir_y == 1 || dir_y == 0)
{
  start_y = 0;
  end_y = height_;
  step_y = 1;
}
else //(dir_y == -1)
{
  start_y = height_ - 1;
  end_y = -1;
  step_y = -1;
}*/
```

Figure 2: First attempt of initialization

In this way, the algorithm will scan all the pixels in the image but won't consider all the pixels in the 0 column and all the ones in the 0 rows, as the first pixels of their path.
That's because the ***pw_*** variables are initialized in a way to consider the image without an external frame of one pixel.
The reason behind this choice is that the census transform and the hamming distances were calculated with a window of size 3x3 and so, to center it correctly, the external border wasn't considered.

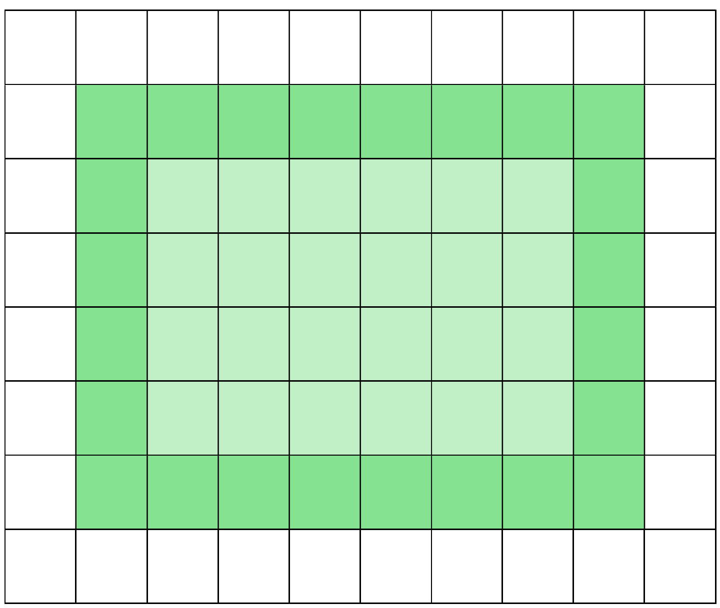


Figure 3: Portion of the image considered by pw_ variables
(the darker cells are the ones that the pw_ variables consider as first)

With this modification, the compute_path_cost will try to calculate the smooth term and will launch *segmentation fault,* because the algorithm tries to access a previous pixel (***q***) that doesn't exist.

To solve this problem, I've corrected the ***pw_*** variables by subtracting 1 from ***pw_.north*** and ***pw._south.***
I didn't touch the other two ***pw_*** because they already considered the first pixels as such.
By doing so, the if condition will now consider as first pixels also the ones in the 0 column and row without launching any error.
In this implementation of the assignment I've obtained good results (Results 1), even though I was calculating the path cost over some pixels that had no cost volume.

Final implementation

After I realized that the hamming distance was done only on the **green area**, I've corrected the variables initialization and reset the if to the starting conditions by removing the ***– 1***.
So now the implemented initialization of the variables is done as in *Figure 4*.

```
//initialize the variables for x depending on the case
if(dir_x == 1 || dir_x == 0)
{
  start_x = pw_.west;
  end_x = pw_.east;
  step_x = 1;
}
else //(dir_x == -1)
{
  start_x = pw_.east;
  end_x = pw_.west;
  step_x = -1;
}
```

```
//initialize the variables for y depending on the case
if(dir_y == 1 || dir_y == 0)
{
  start_y = pw_.north;
  end_y = pw_.south;
  step_y = 1;
}
else //(dir_y == -1)
{
  start_y = pw_.south ;
  end_y = pw_.north ;
  step_y = -1;
}
```

Figure 4: Final implementation

The corresponding results are in Final results.

Task 3

In this task it's created a vector of pairs (***disparity_pairs***).

In the first element of the pair is saved the ***smallest_disparity*** that has good confidence, normalized by *255/disparity_range*.

In the second element is saved the initial guess disparity at the considered pixel (***mono_.at<uchar>(row, col)***).

The normalization is done to get a disparity image in which each pixel has a value of 255 (white) if it's the closest to the camera and 0 (black) if it's the furthest.

Task 4

In this task it was requested to compute the coefficient $x = [h\ k]^T$ that will be used to scale the mono_ disparity values.
To do so, two Eigen matrices are created: $\boldsymbol{b} = \boldsymbol{d_{sgm}}$ and $\boldsymbol{A} = [\boldsymbol{d_{mono}}, \overline{\mathbf{1}}]$.
In these matrices are stored respectively the good disparities calculated with the SGM and the unscaled initial guess disparity.
So basically the first and second elements of the pairs created in the task 3.

After that, *x* is found using the pseudoinverse for the least squares problem for non-homogeneous systems: $\boldsymbol{x} = \left(\boldsymbol{A^T A}\right)^{-1} \boldsymbol{A^T b}$.

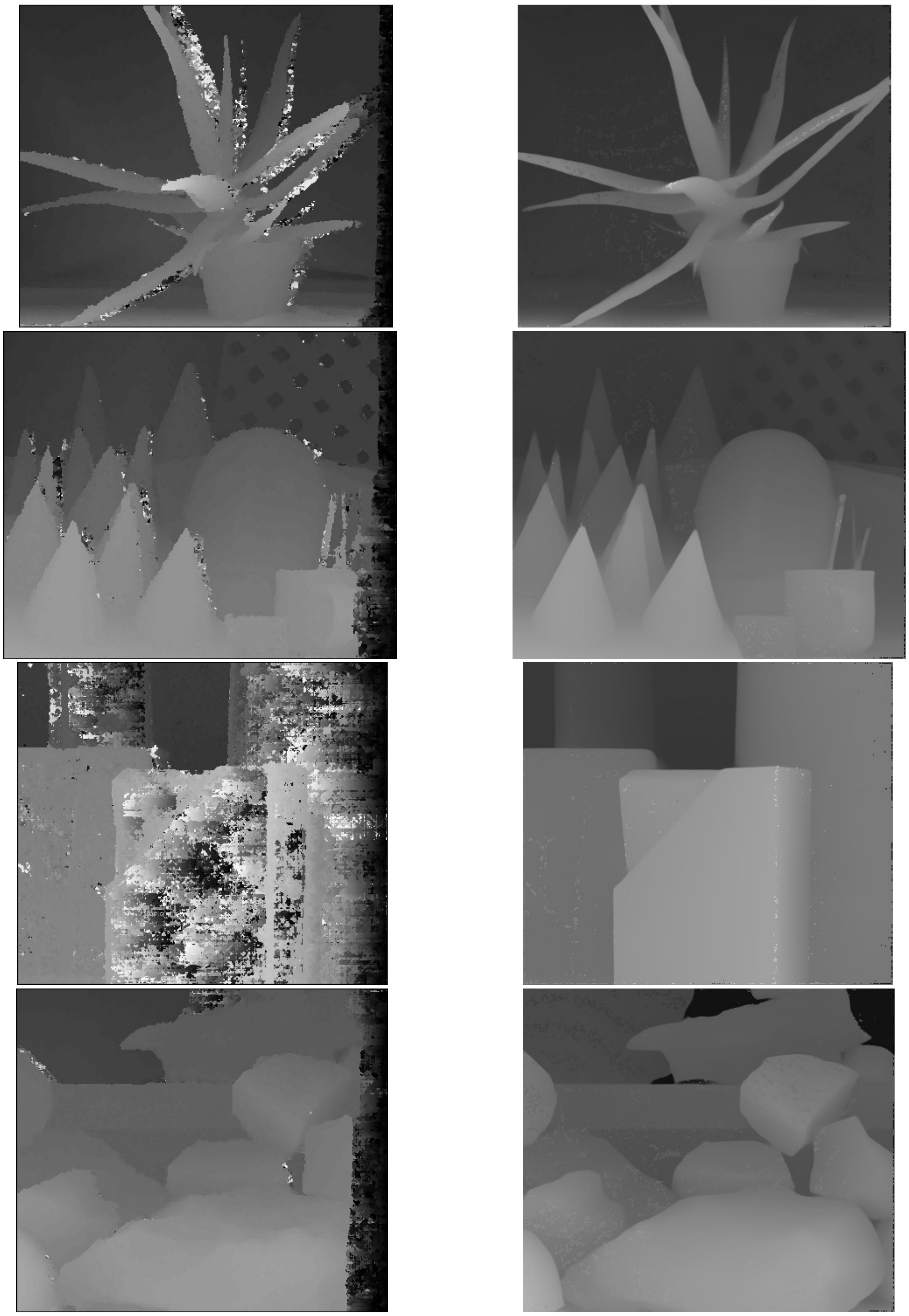
Subsequently, our guesses are scaled and stored in ***disp_***, to complete the previously stored and normalized good $\boldsymbol{d_{sgm}}$.

3. Results

Results 1

	MSE without refining	MSE with refining
Aloe	120.259	13.7148
Cones	467.802	17.4356
Plastic	810.842	343.911
Rocks1	547.745	31.9785

Below are reported the qualitative results
(on the left there are the non-refined disparity and, on the right, the refined ones)



Final results

	MSE without refining	MSE with refining
Aloe	122.464	13.7291*
Cones	475.166	17.4342
Plastic	820.049	348.223
Rocks1	557.735	34.6984

*Here the result changes if I use floating point or double variables, because of rounding errors. 13.7291 for double variables and 13.7283 for float ones.

Below are reported the qualitative results
(on the left there are the non-refined disparity and, on the right, the refined ones)