Report Laboratory 1:

Semi-Global Stereo Matching with Monocular Disparity Initial Guess

Christian Francesco Russo 2087922

Task 1/4: Function `compute_path_cost()`

The function `compute_path_cost()`, given:

- A single pixel $p_i = (\text{cur\_x}, \text{cur\_y})$
- A path with index $\text{cur\_path} \in \{0,1, \dots, \text{PATH\_PER\_SCAN}\}$
- The direction increments $(\text{direction\_x}, \text{direction\_y})$ associated with the path

computes the path cost for p_i for all the possible disparities $d \in [0, \dots, \text{disparity\_range}]$, i.e., the path cost for one column of the integration matrix, which is stored inside the tensor `path_cost_`.

To compute this path cost, by the theory, we need to exploit the following formula:

$$E(p_i, d) = E_{data}(p_i, d) + E_{smooth}(p_i, p_{i-1}) - \min_{0 \le \Delta \le d_{max}} E(p_{i-1}, \Delta)$$

where:

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

In practice, we need to handle two cases:

- If p_i is the first pixel for the given path and for the given direction, i.e., if it is a pixel on the border of the image, then computing the path cost corresponds to compute the first column of an integration matrix, so we have that:
$$E(p_0, d) = E_{data}(p_0, d), \quad \forall d \in [0, \dots, \text{disparity\_range}]$$
since in this case the smooth term is 0, where the term $E_{data}(p_0, d)$ must be extracted from the precomputed tensor cost_.
- If p_i is not a border pixel for the given path and for the given direction, computing the path cost corresponds to compute a column of the integration matrix different from the first one, so:
    - We need to compute the minimum path cost value for the previous pixel p_{i-1} among all possible disparities $d \in [0, \dots, \text{disparity\_range}]$:
    $$\text{best\_prev\_cost} = \min_{0 \le \Delta \le d_{max}} E(p_{i-1}, \Delta)$$
    - For all possible disparities $d \in [0, \dots, \text{disparity\_range}]$, we need to compute:
        - The data term:
        $$\text{no\_penalty\_cost} = E_{data}(p_i, d)$$
        - The first smoothing term:
        $$\text{prev\_cost} = E(p_{i-1}, d)$$
        - The second smoothing term:

$$\text{small\_penalty\_cost} = \min\{E(p_{i-1}, d-1), E(p_{i-1}, d+1)\} + p_{1\_}$$

where we have to handle the cases $d = 0$ and $d = \text{disparity\_range} - 1$.

- The third smoothing term:

$$\text{big\_penalty\_cost} = \ \text{best\_prev\_cost} + p_{2\_}$$

- The smoothing cost can then be computed as:

$$\text{penalty\_cost} = \min\{\text{prev\_cost}, \text{small\_penalty\_cost}, \text{big\_penalty\_cost}\}$$

- The path cost is then:

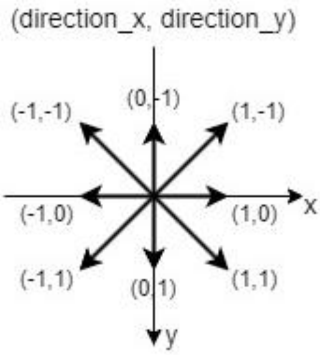
$$E(p_i, d) = \text{no\_penalty\_cost} + \ \text{penalty\_cost} - \ \text{best\_prev\_cost}$$

Task 2/4: Function `aggregation()`

The function `aggregation()` computes the path cost (using the function `compute_path_cost()`) for all pixels in the image, along all possible directions.

To initialize the variables start_x, start_y, end_x, end_y, step_x, and step_y that are used to define the cycles where `compute_path_cost()` is called, we first need to understand how direction works in this program.

The image below summarizes the relation between directions and paths in the program:



paths

element	dx	dy
~~0~~	~~0~~	~~0~~
1	0	-1
2	0	1
3	-1	0
4	-1	-1
5	-1	1
6	1	0
7	1	-1
8	1	1

Notice that the struct $pw\_$ contains the variables initialized as follows: $N = \text{north} = 1, S = \text{south} = h - 1, W = \text{west} = 1, E = \text{east} = w - 1$, that must be used to define the start $= (\text{start\_x}, \text{start\_y})$ and the end $= (\text{end\_x}, \text{end\_y})$ pixels in the cycles.

Case by case, we have the following initializations:

- If $dx = 0 \rightarrow$ vertical direction:

  $$\text{start\_x} = W, \quad \text{end\_x} = E + 1, \quad \text{step\_x} = 1$$

  or equivalently:

  $$\text{start\_x} = E, \quad \text{end\_x} = W - 1, \quad \text{step\_x} = -1$$

  Note: these two initializations are equivalent since the order in which we scan the pixels horizontally does not matter in this case.
- If $dx = -1 \rightarrow$ right-to-left direction:

  $$\text{start\_x} = E, \quad \text{end\_x} = W - 1, \quad \text{step\_x} = -1$$

- If $dx = 1 \rightarrow$ left-to-right direction:

  $$\text{start\_x} = W, \quad \text{end\_x} = E + 1, \quad \text{step\_x} = 1$$

- If $dy = 0 \rightarrow$ horizontal direction:

  $$\text{start\_y} = N, \quad \text{end\_y} = S + 1, \quad \text{step\_y} = 1$$

  or equivalently:

$$\text{start\_y} = S, \ \ \text{end\_y} = N - 1, \ \ \text{step\_y} = -1$$

Note: these two initializations are equivalent since the order in which we scan the pixels vertically does not matter in this case.

- If $dy = -1 \rightarrow$ upward direction:

$$\text{start\_y} = S, \ \ \text{end\_y} = N - 1, \ \ \text{step\_y} = -1$$

- If $dy = 1 \rightarrow$ downward direction:

$$\text{start\_y} = N, \ \ \text{end\_y} = S + 1, \ \ \text{step\_y} = 1$$

The tricky part here was to understand that these variables just define the way the pixels in the image are explored by the two inner cycles, according to the current path. In particular, in general $\text{step\_x} \neq \text{dir\_x}$ and $\text{step\_y} \neq \text{dir\_y}$, indeed dir_x and dir_y, with cur_path, are the variables used to define the orientations and the paths in the external cycle.

Note: in end_x and end_y we add a ± 1 in order to deal with the termination conditions of the two inner cycles: $y \,! = \text{end\_y}$ and $x \,! = \text{end\_x}$.

Note: the last row and the last column of the image are never processed due to how the variables in the struct pw_ are initialized by the program.

Task 3/4: Function `compute_disparity()`

This part is quite simple, we just need to store in a vector of pairs of float each "good" disparity d_{sgm} (i.e., each disparity that satisfies the "if condition"), estimated with SGM, with its corresponding unscaled initial guess disparity d_{mono}, from the mono_ image provided.

Note: d_{sgm} and d_{mono} must be stored as float, because of the computations we have to do in the task 4/4.

Task 4/4: Function `compute_disparity()`

In the function `compute_disparity()` we need to use the disparity pairs, accumulated in the previous task, to estimate the unknown scaling factor and to scale the initial guess disparities accordingly. Finally, we need to use them to improve/replace the low-confidence SGM disparities.

To do that we need to:

- Compute $b = [d_{sgm}]$ and $A = [d_{mono} \ \ 1]^T$.
- Solve the least-squares problem for the nonhomogeneous system $Ax = b$, in order to find $x = [h \ \ k]^T$:

$$x = (A^T A)^{-1} A^T b$$

- Extract the linear scale coefficient h and k from x.
- Scale the initial guess disparities to improve/replace the low confidence SGM disparities.
  To do this we need to iterate all pixels in the image and to use the negation of the previous "if condition".
  When the negated "if condition" is true:

- We compute the scaled initial guess disparity, and we replaced it to the current low confidence SGM disparity as follows:

$$d_{sgm} = h * d_{mono} + k$$

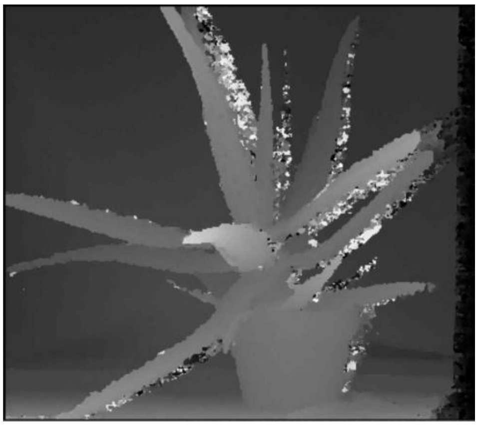
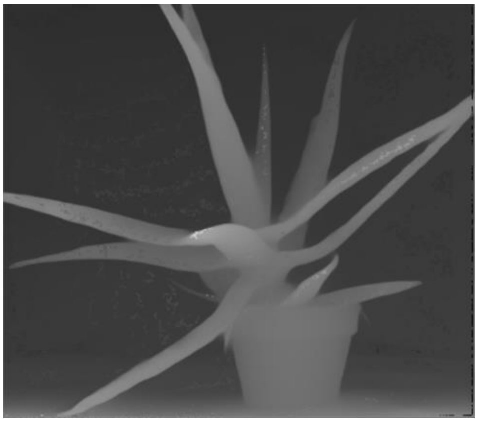
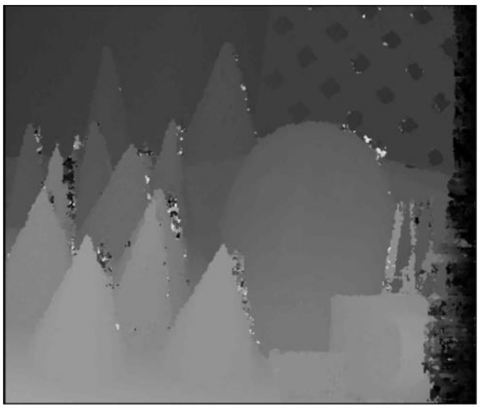
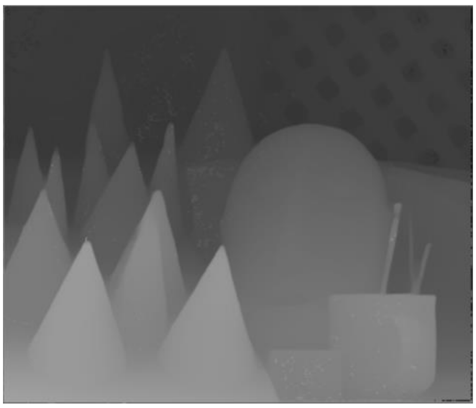
<u>Note</u>: the only tricky part here was to correctly negate the “if condition” applying the De Morgan law properly.

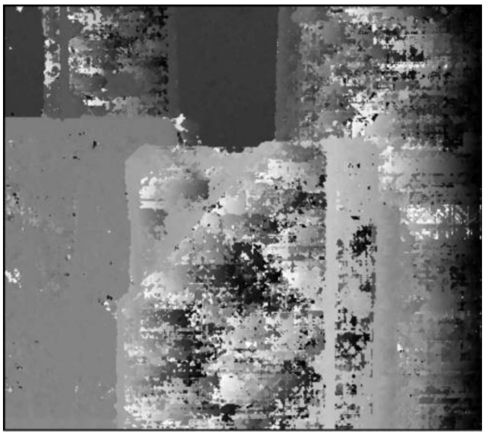
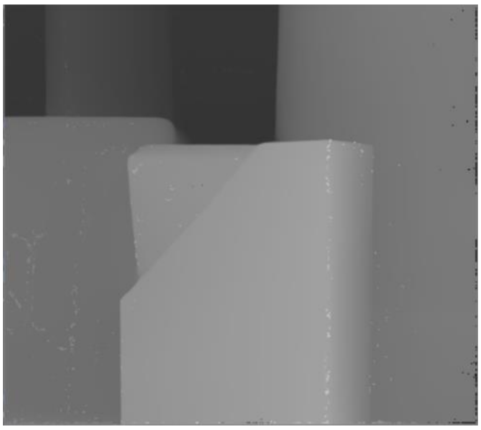
Results

Table 1 - MSE errors without/with refinement, disparity value 85

Data Item	**Aloe**	**Cones**	**Plastic**	**Rocks1**
MSE without refinement	122.464	475.166	820.049	557.735
MSE with refinement	13.7283	17.4342	348.181	34.6984

Table 2 - Images without/with refinement, disparity value 85

Data Item	**Image without refinement**	**Image with refinement**
Aloe		
Cones		

Plastic		
Rocks1