Edoardo Bastianello

ID: 2053077

REPORT LAB 1 – SEMI-GLOBAL MATCHING

Function *compute_path_cost()*:

According to the theory, for a given direction, for a pixel p_i and disparity d:

$$\boldsymbol{E(p_i, d)} = E_{data}(p_i, d) + E_{smooth}(p_i, p_{i-1}) - min_{over\ all\ disparity\ values\ \Delta}\, E(p_{i-1}, \Delta)$$

First, if the current pixel to be processed is the first, for each possible disparity value, I set the path cost relative to the current direction, pixel and disparity equal to the value of the cost volume relative to the same pixel and disparity. In fact, from the theory:

$$E(p_0, d) = E_{data}(p_0, d)$$

If the current pixel is not the first, I found the best path cost of the previous pixel with respect to the current direction over all possible disparity values. This corresponds to the third term of the sum of $\boldsymbol{E(p_i, d)}$ (in the code it's named *best_prev_cost*):

$$min_{over\ all\ disparity\ values\ \Delta}\, E(p_{i-1}, \Delta)$$

Instead, the first term of the sum of $\boldsymbol{E(p_i, d)}$, which is $E_{data}(p_i, d)$, is just equal to the relative value in the cost volume *cost_* for each disparity value *d*.

Next, I had to compute all the three components of $E_{smooth}(p_i, p_{i-1})$, which is the second term of the sum of $\boldsymbol{E(p_i, d)}$, in order to find the minimum between them.
In fact, $E_{smooth}(p, q)$ is equal to the minimum between the following three terms:

1) $E(q, f_q)$ $\quad if\ f_p = f_q$
2) $E(q, f_q) + c_1$ $\quad if\ |f_p - f_q| = 1$
3) $min_{over\ all\ disparity\ values\ \Delta}\, E(q, \Delta) + c_2(p, q)$ $\quad if\ |f_p - f_q| > 1$

Where $f_p$is the disparity of *p* and f_q is the disparity of *q*.
(Note: in the code, c_1 corresponds to the variable $p_{1\_}$, $c_2(p, q)$ corresponds to $p_{2\_}$, $E(q, f_q)$ corresponds to *no_penalty_cost*, $E(q, f_q) + c_1$ corresponds to *small_penalty_cost*, and $min_{over\ all\ disparity\ values\ \Delta}\, E(q, \Delta) + c_2(p, q)$ corresponds to *big_penalty_cost*.)

To compute these three terms, for each disparity, I did the following computations:

- I set $E(q, f_q)$ equal to the path cost of the previous pixel with respect to the current direction.
- For $E(q, f_q) + c_1$, instead, I distinguished between the three following cases:
    - The current disparity is 0: in this case I set the cost equal to the path cost relative to the previous pixel and to the current disparity value+1. This is due to the fact that, since $|f_p - f_q| = 1$, the only one possible correspondence with the previous pixel is relative to the next disparity value (which is 1). Then, I added the penalty $p_{1\_}$ to the cost.
    - The current disparity is equal to its max possible value: in this case I set the cost equal to the path cost relative to the previous pixel and to the current disparity value-1, since this is the only possible correspondence (similarly to the previous case). Then, I added the penalty factor $p_{1\_}$.
    - All the other possible disparity values: in this case there are two possible correspondences with the previous pixel since $|f_p - f_q| = 1$: its path cost relative to the current disparity-1

and its path cost relative to the current disparity+1. I computed both and kept the minimum, after adding the penalty factor $p_{1\_}$.

- As for $min_{over\ all\ disparity\ values\ \Delta}\ E(q, \Delta) + c_2(p, q)$, instead, I simply set it equal to *best_prev_cost,* which I computed earlier, and I added the big penalty factor $p_{2\_}$.
- After computing all these three terms, I assigned to $E_{smooth}(p, q)$ the value of the minimum between them.

Finally, for all disparity values, I computed the path cost relative to the current path, to the current pixel *p* and to the current disparity *d* as the following sum of terms, as stated at the beginning of the report:

$$\boldsymbol{E(p_i, d)} = E_{data}(p_i, d) + E_{smooth}(p_i, p_{i-1}) - min_{over\ all\ disparity\ values\ \Delta}\ E(p_{i-1}, \Delta)$$

Function *aggregation()*:

In order to initialize the variables *start_x, start_y, end_x, end_y, step_x, step_y* with the right values, I checked the value of *dir_x* and *dir_y*. In this way, I had the following cases:

- Direction from left to right with respect to the x direction (case *dir_x=1*): in this case, I set *start_x* to *pw_.west* since this is the leftmost coordinate. For *end_x*, instead, I set it equal to *pw_.east+1*, since this is the rightmost coordinate (the +1 is due to the fact that in the for cycle in the next section of the code, the last value is excluded from the condition of the cycle (*x != end_x*)). Finally, I set *step_x=1* to scan each pixel of the image (left to right).
- Direction from right to left with respect to the x direction (case *dir_x=-1*): this is the opposite case with respect to the previous one, so I set *start_x* to the *pw_.east*, *end_x* to the *pw_.west–1* (the -1 is due to the same reason as the +1 of the previous case) and *step_x* to -1 to scan all the pixels of the image from right to left.
- Vertical path (case *dir_x=0*): in this case I set *start_x=pw_.west, end_x = pw_.east+1* and *step_x=1*. I made this choice so that all possible vertical paths are scanned (from left to right). Note that, in this case, I could have set *start_x=pw_.east*, *end_x=pw_.west-1* and *step_x=–1* instead, since the order in which I scan the pixels with respect to the x axis doesn't matter.
- Direction from top to bottom with respect to the y direction (case *dir_y=1*): in this case, I set *start_y* equal to *pw_.north* since this is the top pixel. For *end_y*, instead, I set it equal to *pw_.south+1*, since this is the bottom pixel (as in the previous cases, the +1 is just to balance the fact that the last term will be excluded from the for cycle). Finally, I set *step_x=1* to scan each pixel of the image from the top to the bottom.
- Direction from bottom to top with respect to the y direction (case *dir_y=-1*): in this case, I set *start_y* equal to *pw_.south*, *end_y* equal to *pw_.north-1*, and *step_x equal to -1* to scan each pixel of the image from the bottom to the top.
- Horizontal path case (*dir_y=0*): similarly to the case of vertical path, I set *start_y=pw_.north, end_y=pw_.south+1, step_y=1*. In this way all the horizontal paths of the image are scanned (from top to bottom). Note that, similarly to the case of dir_x=0 (vertical path) I could have set *start_y=pw_.south, end_y=pw_.north-1, step_y=-1* since, in this case, the order in which I scan the pixels with respect to the y axis doesn't matter.

To aggregate the costs for all direction into the *aggr_cost_* tensor:

- I scanned all pixels and all disparity values.
    - For each possible direction, I updated the value of the *aggr_cost_* tensor by adding the relative value of *path_cost* while at the same time removing its component $E_{data}(p, d)$ (which is the value in the cost volume *cost_*). I removed this component to prevent it from being added up multiple times (one time for each possible path).

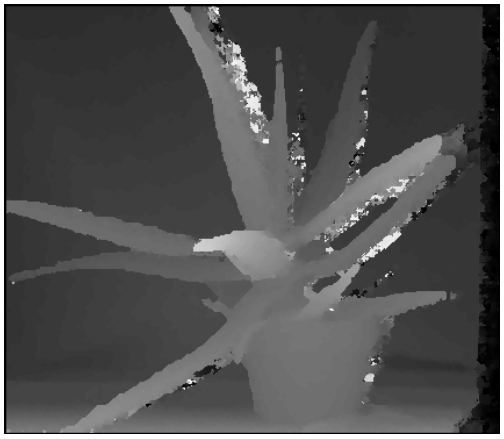
- Then, I added back to *aggr_cost_* one $E_{data}(p, d)$ component.

By doing so, for each possible pixel and for each possible disparity, I aggregated the *path_cost* of each direction.

Results:

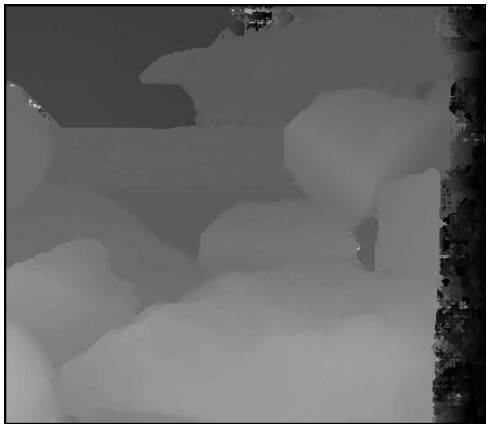
Aloe

Right image MSE: 106.987



Rocks1

Right image MSE: 346.657



Cones

Right image MSE: 470.417