3D DATA PROCESSING - LAB 1

Topic: Semi Global Stereo matching.
Goal: Disparity maps estimation of stereo images.

Extend the provided C++ software implementing the computation of the path costs and the following cost aggregation.

The provided software already implements the following methods:

- `SGM(...)`
  - define the parameters to be used during the computation like window size for Hamming distance, disparity range and penalty factors.
- `void set(...)`
  - loading of input images.
  - initialization of class parameters.
- `void calculate_cost_hamming()`
  - computation of cost volume using Census transform and Hamming distance between patches.
- `void init_paths()`
  - initialize the directions for each path.

The goal is to extend the **`compute_path_cost()`** and **`aggregation()`** methods on **sgm.cpp** in order to successfully perform disparity estimation.

In the function **`aggregation()`** variables **`start_x`**, **`start_y`**, **`end_x`**, **`end_y`**, **`step_x`** and **`step_y`** are used to define the cycles where **`compute_path_cost()`** is called.
To define the starting and end points, use the **`pw_`** structure that holds the minimum and maximum horizontal and vertical coordinates.
Be careful to instantiate the values of `step_x` and `step_y` in order to scan the cost volume in the correct order (hint: check the value of dir_x and dir_y, for instance dir_x = -1 and dir_y = 0 means a right to left horizontal path).

```
void SGM::aggregation()
{
 //for all defined paths
 for(int cur_path = 0; cur_path < PATHS_PER_SCAN; ++cur_path)
 {

   int dir_x = paths_[cur_path].direction_x;
   int dir_y = paths_[cur_path].direction_y;

   int start_x, start_y, end_x, end_y, step_x, step_y;

   //TO DO: initialize the variables start_x, start_y, end_x, end_y,
next_dim_x, next_dim_y with the right values
```

```
 }
//TO DO: aggregate the costs for all direction into the aggr_cost_ tensor
}
```

The **compute_path_cost()** function, given a point p defined by its coordinates **cur_x** and **cur_y** and a path with index **cur_path** and direction defined by **direction_x** and **direction_y** (both can be -1, 0, or 1), should compute the path cost for p for all the possible disparities d from 0 to disparity_range (excluded). The output should be stored in **path_cost_[cur_path][cur_y][cur_x][d]**, for all possible d. The matching cost (data term) should be recovered from the the cost volume, e.g. **cost_[cur_y][cur_x][d]**;
To update the path cost remember to use the class variables **p1_** and **p2_** which represent respectively the small and big penalty added factors.

```
void SGM::compute_path_cost(int direction_y, int direction_x, int cur_y, int
cur_x, int cur_path)
{
 //use this variables if needed
 unsigned long prev_cost;
 unsigned long best_prev_cost;
 unsigned long no_penalty_cost;
 unsigned long penalty_cost;
 unsigned long small_penalty_cost;
 unsigned long big_penalty_cost;

 // if the processed pixel is the first:
 if(cur_y == pw_.north || cur_y == pw_.south || cur_x == pw_.east || cur_x ==
pw_.west)
 {
   //Please fill me!
 }

 else
 {
   //Please fill me!
 }
}
```

ADDITIONAL HINTS:

To use:

- **cost_**, cost volume precomputed by matching (right to left pixels, for all possible disparities) census descriptors using hamming distances, over a support window. It is defined, for each possible pixel x, y and possible disparity d (e.g. cost_[y][x][d]). Total size is (height_, width_, disparity_range_)

- **`path_cost_`**, current right cost for each path, updated after each iteration of `compute_path_cost()`. It is defined for each possible path i, pixel x, y and possible disparity d (e.g. path_cost_[i][y][x][d]). Total size is (PATHS_PER_SCAN, height_, width_, disparity_range_)

- **`aggr_cost_`**, current right cost after aggregation of costs over all the defined paths. It is defined, for each possible pixel x, y and possible disparity d (e.g. aggr_cost_[y][x][d]). Total size is (height_, width_, disparity_range_)

- **`pw_`**, structure that holds the minimum and maximum horizontal (east and west) and vertical (north and south) coordinates.

- **`paths_`**, vector containing the all paths encoded by their scan direction (for example direction_x = -1 and direction_y = 0 means a right to left horizontal path, east to west).

SAMPLE OUTPUT,