3D DATA PROCESSING - LAB 1

Topic: Stereo matching
Goal: Disparity maps estimation of stereo images using Patch Matching algorithm.

Extend the provided C++ software with the patch match core functionalities: **disparity propagation** and **random search**.

The provided software already implements the following methods:

- `void set(...)`
  - loading of input images
  - loading of precomputed cost volumes
  - random disparity maps initialization.
- `void process(...)`
  - forward and backward scanning of left and right views
- `void postProcess()`
  - post-processing of estimated disparity maps with left/right consistency check, filtering, and smoothing

Disparities are stored inside the disps_ matrices, while current costs are stored inside the costs_ with the same size as the input images (stored in views_ matrices):

- `cv::Mat1f views_[2]`, current left and right images
- `cv::Mat1f disps_[2]`, current left and right disparity maps
- `cv::Mat1f costs_[2]`, current left and right costs

(The 0 index refers to the left image or the left-to-right disparity/cost).

The goal is to extend the `process(...)` method to perform (in order):

1) Spatial propagation
2) Random search around the current disparity
3) View propagation

This should be implemented by completing the three methods:

- `spatial_propagation()`
- `disp_perturbation()`
- `view_propagation()`

called by the provided `process_pixel()` method which is called by the already implemented `process()` method:

```
/* x,y pixel coordinates, cpv index of base view (i.e., 0 for
 * left-to-right process, 1 for right-to-left process), iter is
 * the iteration id:
 * -if iter is even then the function check the left and
 * upper neighbours
 * -if iter is odd then the function check the right and lower
 * neighbours */

void PatchMatch::process_pixel(int x, int y, int cpv, int iter)
{
  // spatial propagation
  spatial_propagation(x, y, cpv, iter);

  // disparity refinement
  disp_perturbation(x, y, cpv, MAX_DISPARITY / 2, .5f);

  // view propagation
  view_propagation(x, y, cpv);

}


void PatchMatch::process(int iterations)
{
  for (int iter = 0 ; iter < iterations; ++iter)
  {
    bool iter_type = (iter % 2 == 0);
    // PROCESS LEFT AND RIGHT VIEW IN SEQUENCE
    for (int work_view = 0; work_view < 2; ++work_view)
    {
      if (iter_type)
      { // FORWARD SCANNING
        for (int y = 0; y < rows_; ++y)
          for (int x = 0; x < cols_; ++x)
            process_pixel(x, y, work_view, iter);
      }
      else
      { // BACKWARD SCANNING
        for (int y = rows_ - 1; y >= 0; --y)
          for (int x = cols_ - 1; x >= 0; --x)
            process_pixel(x, y, work_view, iter);
      }
    }
  }
}
```

Note:

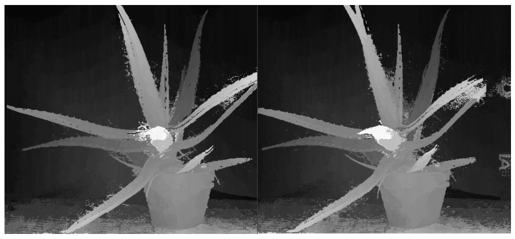
- The disparities range is between -60 and 0 for the left-to-right case and between 0 and 60 for the right-to-left case.

- Use the `float precomputed_disp_match_cost(...)` method to get the precomputed matching cost of a patch centered at (x, y) in the current view with the patch centered at (x+disparity, y) in the other view. Note that `precomputed_disp_match_cost()` requires a positive disparity also for the left-to-right case, since the sign is automatically managed internally. (This is not the case when dealing with the view propagation step).

Example:

```
{
  ...

  // get current left disparity at location (x,y)
  float current_disp = disps_[0](y, x);

  // get current left matching cost at location (x,y)
  float current_cost = costs_[0](y, x);

  // perturb the current disparity and test it
  float new_disp = current_disp + 10.0f;
  float new_cost = precomputed_disp_match_cost(new_disp, x, y, 0);

  // if new cost is smaller, update left disparity maps and costs
  if (new_cost < current_cost)
  {
    disps_[0](y, x) = new_disp;
    costs_[0](y, x) = new_cost;
  }
}
```

SAMPLE OUTPUT



Dataset

The dataset containing three samples can be downloaded from [here]().

Project delivery

The project must be developed in C++ with the OpenCV library. Your code must compile smoothly (i.e. without errors) within Ubuntu Linux 20.04 or similar distributions. No building systems other than CMake are allowed. At the end of the compilation, the code should be tested with the already implemented executable:

```
./patchmatch path/to/dataset
```

where `path/to/dataset` is one between Aloe, Cones, Rocks1 (already provided):

You need to deliver a single archive (tar.gz. zip,...) including:

- All the source code (**without** object files, executables, datafile and datasets):
- A brief report that shows the obtained disparities in the Aloe, Cones, Rocks1 datasets along with the numerical results provided in output by the `patchmatch` application for each dataset.