Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Semi-Global Matching
Result & Report

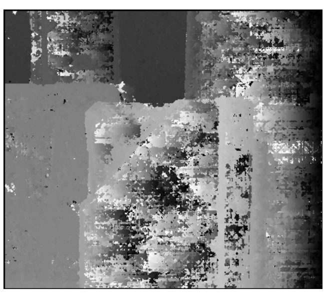
Francesco Crisci 2076739

April 19^{th} 2024

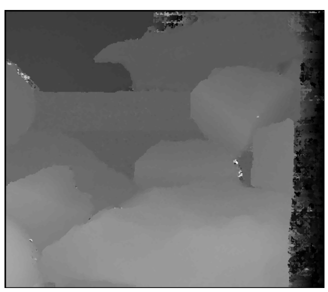
Results

To begin with, here I'm gonna show the results obtained before and after the refinement procedure.

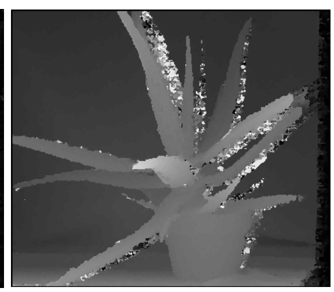
- **Results before refinement:**



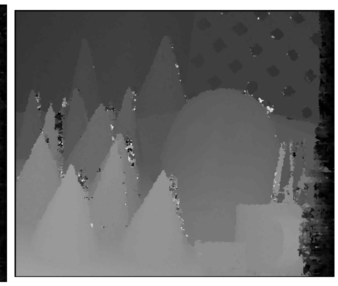
Plastic MSE: 820.049



Rocks MSE: 557.735

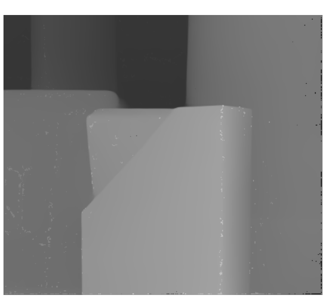


Aloe MSE: 122.452

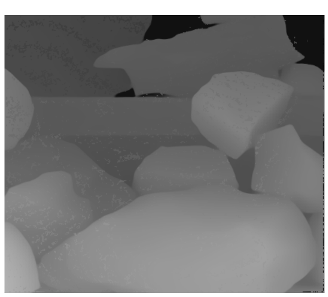


Cones MSE: 475.159

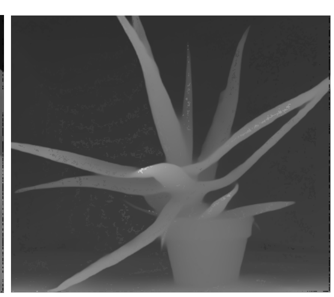
- **Results after refinement:**



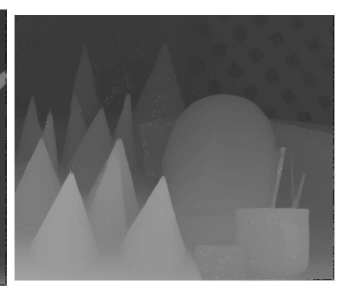
Plastic MSE: 348.223



Rocks MSE: 34.6984



Aloe MSE: 13.7204



Cones MSE: 17.382

Implementation

Calculate Path Cost

The function `compute_path_cost()` can be decomposed in two phases:

I The first part checks if the pixel we are processing is the first one. If that is the case, we use the following equation:

$$E_{smooth}(p, q) = E(q, f_q)$$

where f_q is the disparity level of pixel q.

II The second part computes the smoothness cost of the current pixel with the following equation:

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

Aggregation

In the function `aggregation()` I've initialized the variables according to the desired direction and then allowed the computation of the path cost. The reasoning has been the following:

- If the x has direction equal to -1, we want to go from east to west with a step of -1.
- If the x has direction equal to +1, we want to go from west to east with a step of +1.
- If the y has direction equal to -1, we want to go from south to north with a step of -1.
- If the y has direction equal to +1, we want to go from north to south with a step of +1.

Compute Disparity

In the `compute_disparity()` I've implemented the following:

I In the first part, we store the good disparities from the initial guess and from the disparity computed in the previous steps.

II Then, in the second part, using the solution for non homogeneous systems, I've retrieved the h and k factors of the following equation:

$$d_{sgm} = h \times d_{mono} + k$$

Once Obtained the values, I've scaled and improved only the values that had a small confidence.

Code Snippets

```cpp
if (cur_y == pw_.north || cur_y == pw_.south || cur_x == pw_.east || cur_x == pw_.west) {
    //Please fill me!
    for (unsigned int d = 0; d <disparity_range_; d++) {
        path_cost_[cur_path][cur_y][cur_x][d] = cost_[cur_y][cur_x][d];
    }
} else {
    //Please fill me!
    best_prev_cost = path_cost_[cur_path][cur_y-direction_y][cur_x-direction_x][0];
    for(int d = 0; d < disparity_range_;d++){
        prev_cost = path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][d];
        no_penalty_cost = prev_cost;
        if (d == 0) {
            small_penalty_cost = path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][d + 1] + p1_;
        } else if (d == disparity_range_ - 1) {
            small_penalty_cost = path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][d - 1] + p1_;
        }else {
            if (path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][d + 1] >=
                path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][d - 1])
                small_penalty_cost = path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][d - 1] + p1_;
            else
                small_penalty_cost = path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][d + 1] + p1_;
        }
        best_prev_cost = 10000;
        for (unsigned int dd = 0; dd < disparity_range_; dd++) {
            if (int(dd - d) >= 2 || int(dd - d) <= -2) {
                if (best_prev_cost > path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][dd])
                    best_prev_cost = path_cost_[cur_path][cur_y - direction_y][cur_x - direction_x][dd];
            }
        }
        big_penalty_cost = best_prev_cost + p2_;
        unsigned long temp = min(no_penalty_cost, small_penalty_cost);
        path_cost_[cur_path][cur_y][cur_x][d] = cost_[cur_y][cur_x][d] + min(temp, big_penalty_cost);
    }
```

CODE TO BE IMPLEMENTED 1/4

```cpp
switch (dir_x)
{
case -1:
    start_x=pw_.east;
    end_x=pw_.west;
    step_x=-1;
    break;
default:
    start_x=pw_.west;
    end_x=pw_.east;
    step_x=1;
    break;
}
switch (dir_y)
{
case -1:
    start_y=pw_.south;
    end_y=pw_.north;
    step_y=-1;
  break;
default:
    start_y=pw_.north;
    end_y=pw_.south;
    step_y=1;
  break;
}
for (int y = start_y; y != end_y; y += step_y) {
    for (int x = start_x; x != end_x; x += step_x) {
        compute_path_cost(dir_y, dir_x, y, x, cur_path);
    }
}
```

CODE TO BE IMPLEMENTED 2/4

```cpp
d_mono.push_back(float(mono_.at<uchar>(row, col)));
d_sgm.push_back(smallest_disparity);
```

CODE TO BE IMPLEMENTED 3/4

```cpp
float h, k;
Mat A, transpose_A, b(d_sgm);
const Mat vector_of_ones = Mat_<float>::ones(1, d_mono.size());
cv::vconcat(d_mono, vector_of_ones, transpose_A);
A = transpose_A.t();
Mat temp_x, temp_x_2, x;
temp_x = transpose_A * A;
temp_x_2 = temp_x.inv() * transpose_A;
x = temp_x_2 * b;
h = x.at<float>(0,0);
k = x.at<float>(0,1);
for (int row = 0; row < height_; ++row) {
    for (int col = 0; col < width_; ++col) {
        if (inv_confidence_[row][col] <= 0 || inv_confidence_[row][col] >= conf_thresh_) {
            disp_.at<uchar>(row,col) = (mono_.at<uchar>(row,col)* h +k)* 255 / disparity_range_;
        }
    }
}
```

CODE TO BE IMPLEMENTED 4/4