**Report – Stefano Deriu - 2078244**

**compute\_path\_cost()**

In the compute\_path\_cost() function I’ve simply followed the equation from the Semi-global matching paper provided in moodle.

In particular for the case in which a pixel is located at the border of the image I’ve considered the the smoothness term to be less reliable so that the path\_cost is equal to the Data term C(p,d).

In all the other cases, I’ve calculated prev\_cost as the path\_cost of the previous pixel along the path direction for all possible disparities.  
The small\_penalty\_cost as the path\_cost taken for the d+1 and d-1 disparities + p1\_ of the previous pixel along the path direction.  
And the big\_penalty\_cost as the minimum of the path\_costs of the previous pixel along the path direction + p2\_.

Then considered only the minimum of these costs and calculated the entire formula.

**aggregation()**

The initialization of the variables follows the values for dir\_x and dir\_y. For example:

dir\_x = 1, dir\_y = -1

start\_x = 0, end\_x = width\_ - 1, step\_x = dir\_x

start\_y = height\_ -1 , end\_y = 0 , step\_y = dir\_y

**compute\_disparity()**

Here I’ve created a struct to store the vectors for the high confidence disparities and initial guesses

struct DisparityPair

{ int d\_mono;

int d\_sgm; }

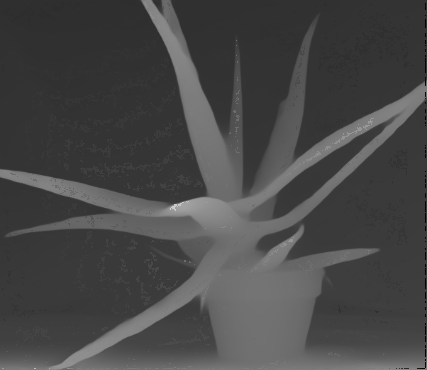
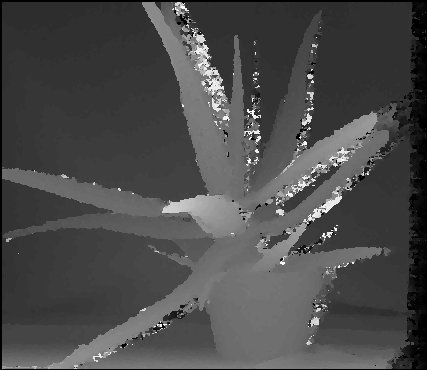
Then through the use of the Eigen library I’ve initialized the matrix A and the vectors b and x.

With the use of .transpose() and inverse() functions I’ve implemented , found h and k from it and then replaced every low confidence disparity with the respective scaled initial guess.

**Quantitative results:**

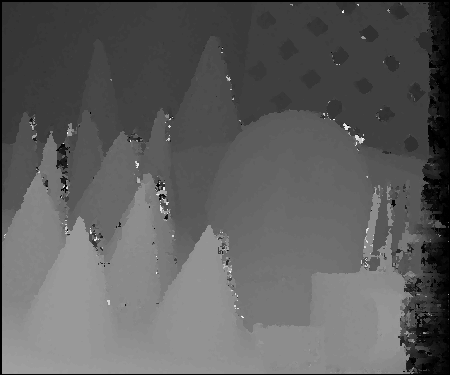
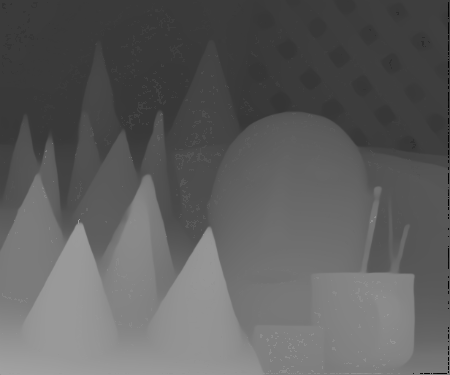
**Aloe**:

MSE with only SGM: 122.464

MSE after refinements: 13.7291

**Cones**:

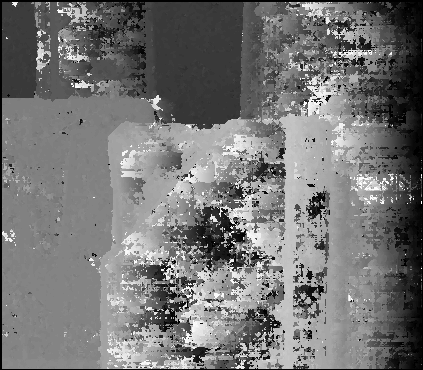
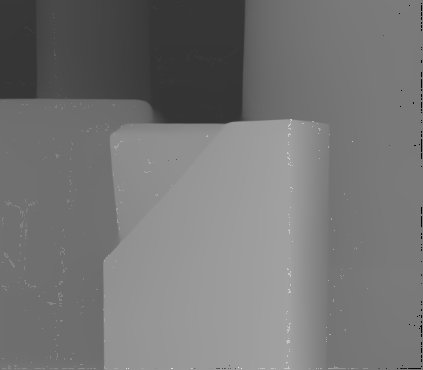
MSE with only SGM: 475.166

MSE after refinements: 17.4342

**Plastic**:

MSE with only SGM: 820.049

MSE after refinements: 348.223



**Rocks**:

MSE with only SGM: 557.735

MSE after refinements: 34.6984

