# Assignment 2 – Stereo Matching

Roee Meir 315297408 & Elad Shadman 211336888

## Part A: Distance Tensor Computation

### Part 1

**Task: Compute ssdd(1,2,2), ssdd(1,2,3), ssdd(2,3,0), ssdd(2,3,1) in a similar way.**

Giving the following toy example:



:

  
We assume the outside is padded by zeros.

:



:



:



### Part 2

Task: Write a function that calculates the tensor of SSD distances between the two images, according to the example above.

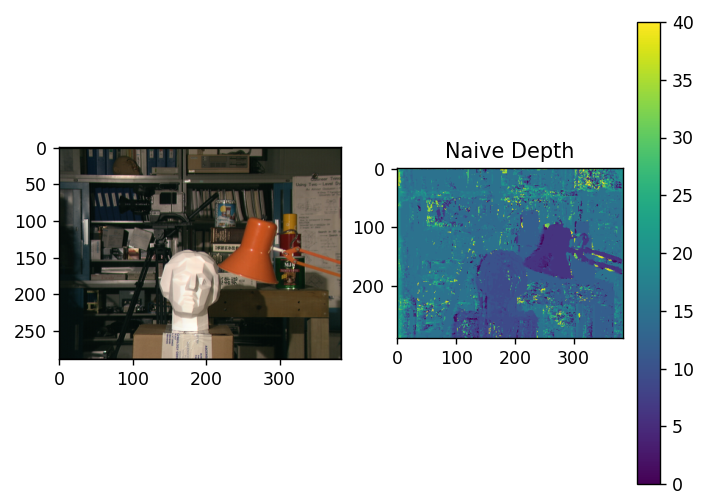
We shift the right image for each possible disparity and compute the squared difference between the two images.  
Then, we use convolution with an all-ones kernel to sum these squared differences over the window instead of looping through every pixel.  
This gives us the SSD value for each pixel at each disparity and forms the SSDD tensor.  
We chose convolution because it performs the window summation efficiently and matches exactly what the SSD calculation requires.  
**Visual Result:**  


When visualizing disparities in , we can see object edges becoming sharp only at the correct disparities and blurry or doubled at incorrect ones.  
This shows how alignment quality changes with disparity and where the left and right images best match.  
Overall, the visuals confirm that the SSD tensor captures depth structure as expected.

## **Part B: Naive Depth Map**

### Part 3

Write a function that builds a depth map out of the SSDD tensor without any smoothing.

**The result:  
**

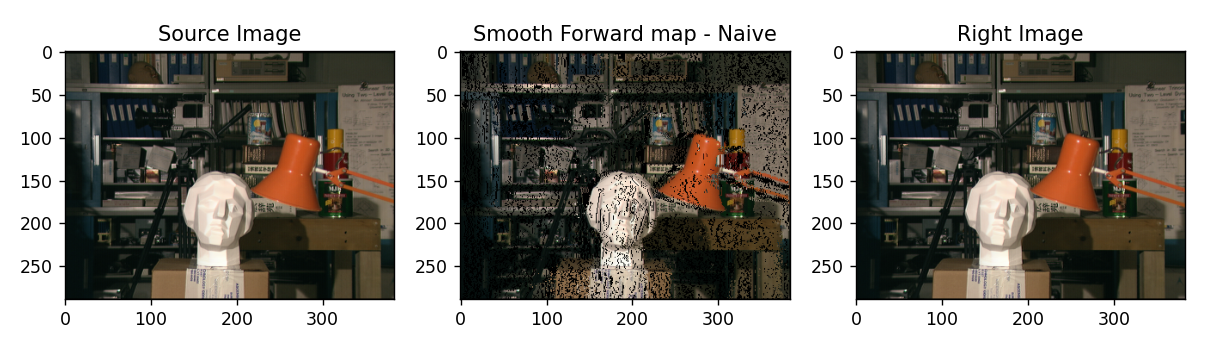
The naive labelling output shows significant noise because each pixel selects its disparity independently, without considering neighbouring pixels.

In addition, the provided implementation of the SSD calculation used an incorrect disparity convention, effectively reversing the correspondence between the left and right images. As a result, the matching process favoured the opposite shift direction, causing close objects to be assigned lower disparity values and distant objects higher ones.

### Part 4

Task: Give a short explanation to the result. Outline the problems of using the SSDD tensor in a naive approach. What are the reasons for the problems you mentioned?

The naive disparity map demonstrates that simply choosing the minimum SSD per pixel is not sufficient for producing a reliable smooth depth estimation.  
Although some object structures are visible, large regions appear noisy due to ambiguity or lack of texture.  
Overall, the result highlights that depth estimation requires spatial smoothing.

**Using forward mapping:**  


When comparing the forward-mapped image to the real right image, we see good alignment only in regions where the naive disparity estimates were correct.  
In texture less areas and along depth boundaries, the images differ significantly, and gaps or distortions appear.

## **Part C: Depth Map Smoothing using Dynamic Programming**

### Part 5

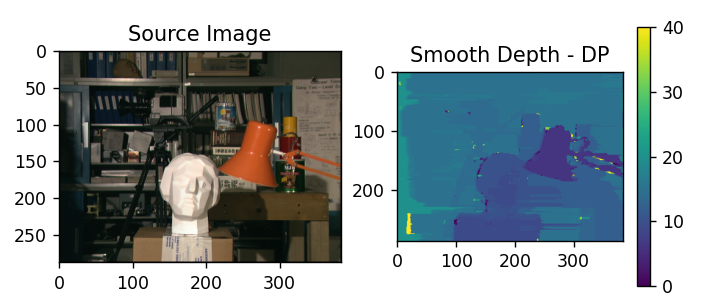
Task: Implement the score method for a single slice of the ssdd tensor, using Dynamic Programming

|  |  |
| --- | --- |
|  |  |
|  |  |

In these DP slices, the x-axis is the image columns, the y-axis is the disparity, and the colour shows the DP cost (dark = low cost, bright = high cost). Around the columns where the statue appears, we see dark “valleys” at higher disparities, because the statue is close and matches best there.  
At row = 150 we also see a darker valley at slightly lower disparities where the lamp is, but at row = 200 the lamp is no longer in that scanline, so only the statue’s low-cost valley remains while the background stays at low disparities with higher costs.

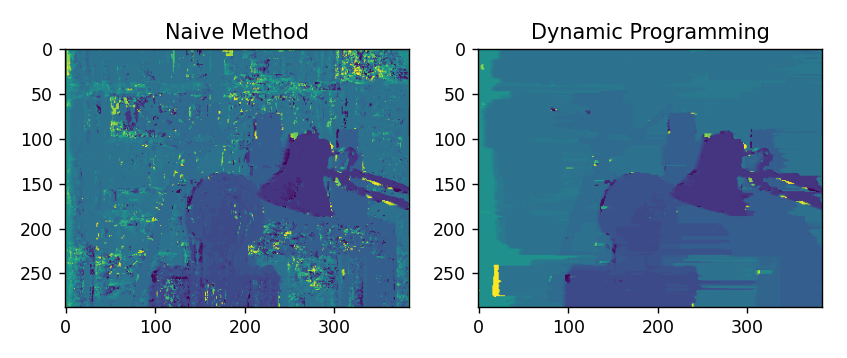
### Part 6

Task: Implement a method which takes the full ssdd tensor and outputs a depth map, using Dynamic Programming.

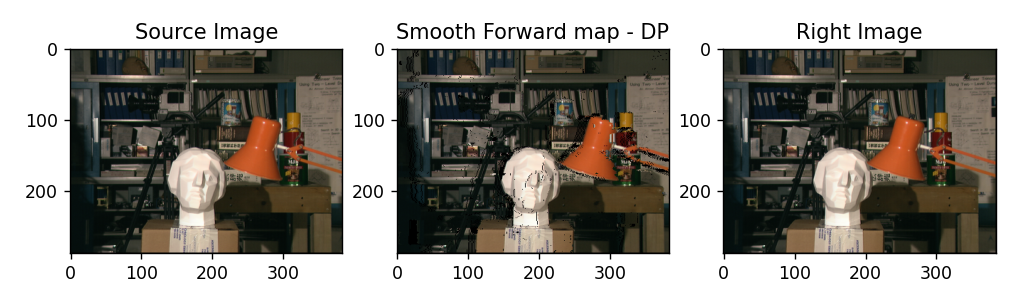


### Part 7

Task: What is the difference between the depth map obtained from the previous item and the map obtained naively? Include the two depth maps in your report.

**Comparing both results:  
**

The naive method produces a very noisy disparity map with many random jumps and broken surfaces because each pixel is labelled independently.  
Dynamic Programming smooths the disparities along each row, removing speckles and making object surfaces like the statue and lamp much more continuous.  
Overall, the DP result better reflects the real depth structure of the scene while preserving important boundaries.

**Using forward mapping:**  


The DP-based forward-mapped image aligns much better with the right image, while the naive method produced many holes due to inconsistent disparities.  
With DP smoothing, foreground objects like the statue and lamp shift to the correct locations, showing that the disparities are more reliable.  
Only a few distortions remain in occluded or low-texture regions.

## Part D: Depth Image Smoothing Using Semi-Global Mapping

## Part 8

Task: Implement a method which extracts slices from the ssdd according to a direction which it receives as an input.

To extract slices in each direction, we represented the image as coordinate pairs and generated 1-D paths by scanning rows, columns, and diagonals.

Main diagonals were obtained using the relation , and anti diagonals using , with validity masks keeping only in-bounds pixels. For each direction, we optionally reversed the order to cover all eight orientations and returned the pixel indices forming each slice.

## Part 9

Task: Along diagonal directions there are slices of shorter lengths so you’ll need to update dp\_grade\_slice to handle slices of shorter lengths.

Diagonal slices have variable length, but our dp\_grade\_slice already computes DP over N columns, so it automatically adjusts to any slice size. Since the algorithm depends only on the slice’s actual width, no code changes were needed.

## Part 10

Task: Implement a method which, given a full ssdd tensor, computes the depth map according to the Semi-Global Mapping approach.  
Given the SSDD tensor , which stores the matching cost for each pixel under disparity label , we computed a refined cost:

For a single direction, we used this cost function:

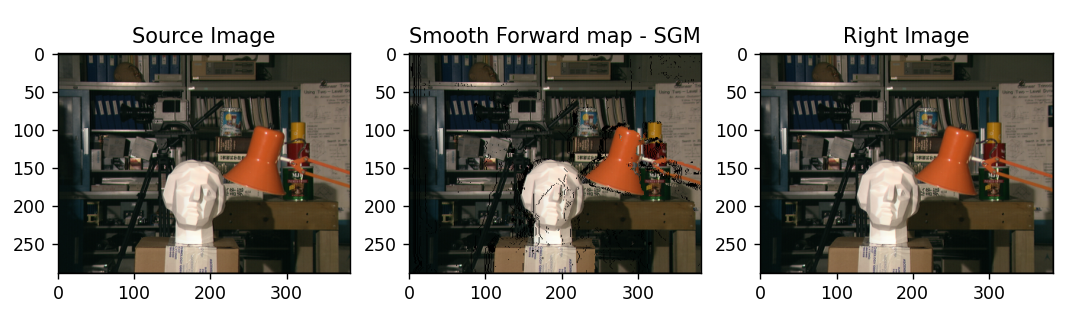
At last, we picked the depth label with minimal aggregated cost:

**The result:**



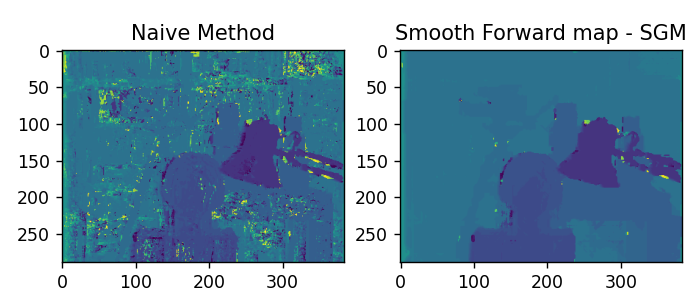
The scene contains large texture less, so local SSDD matching is ambiguous, but SGM encourages neighbouring pixels to agree, producing a stable but smooth output.

**Using forward mapping:**

The closer objects (like the head) align well, while small distortions and gaps reveal occlusions and regions where depth estimation is less certain.

## Part 11

Task: What is the difference between the depth map obtained from the previous item and the map obtained naively? Include the two depth maps in your report.

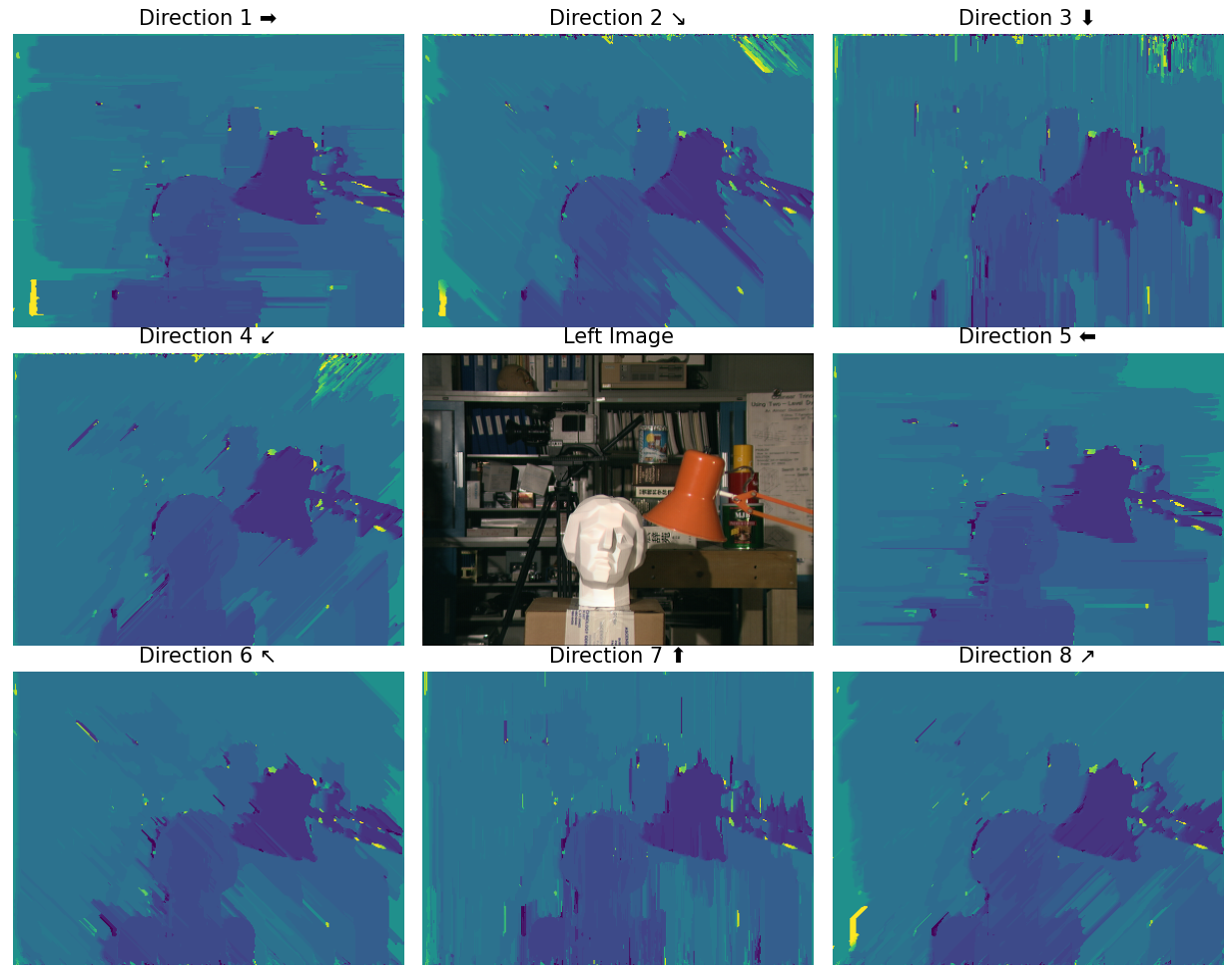
**Compare Naive approach to SGM Smoothing:**

The naive method produces a highly noisy disparity map with strong pixel-level variations, especially in flat or low-texture regions. In contrast, the SGM result is much smoother and more stable because it aggregates information along multiple directions. As a result, object shapes become clearer, noise is reduced, and depth boundaries are preserved far better than in the naive approach.

## Part 12

Task: To debug your result, implement a method which, given a full ssdd tensor, computes the depth map according to the Semi-Global Mapping approach for each direction. This method should return a dictionary mapping each direction (integer: 1, ..., 8) to a depth map computed with an L tensor obtained from that direction.

**The Result:**



The per-direction depth maps illustrate how each path orientation contributes a different type of structural bias: horizontal directions emphasize horizontal consistency, vertical directions enforce vertical smoothness, and diagonal directions capture oblique geometric patterns. Individually, each map contains streaking artifacts aligned with its direction, but together they expose complementary information. This visualization confirms that averaging all eight directions in SGM is essential - only the combined result produces a balanced, stable, and accurate depth map.

# Part E: Your own image

## Part 13

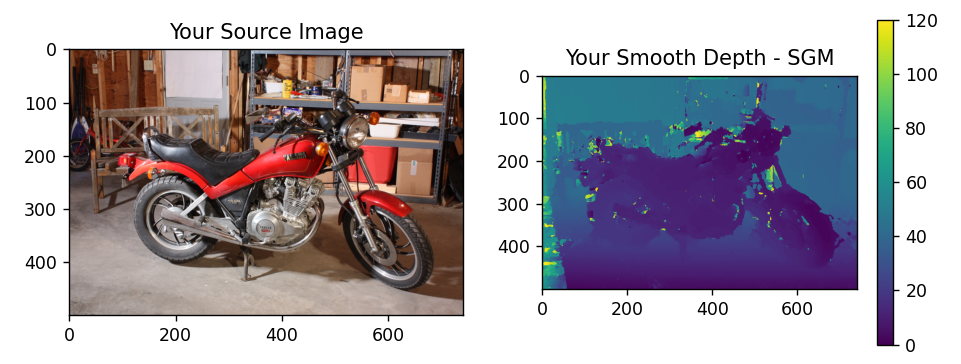
Task: Use a pair of your own images. Make sure the images have been rectified. Run your code which generates the Semi-Global Mapping on the pair of images. We recommend using images from: https://vision.middlebury.edu/stereo/data/ . For the main script to work: name the left image: “my\_image\_left.png” and the right image: “my\_image\_right.png”. You may overrun the parameters: COST1, COST2, WIN\_SIZE and DISPARITY\_RANGE in the “Your image playground section” at the bottom of the script.

**The photos we chose:**

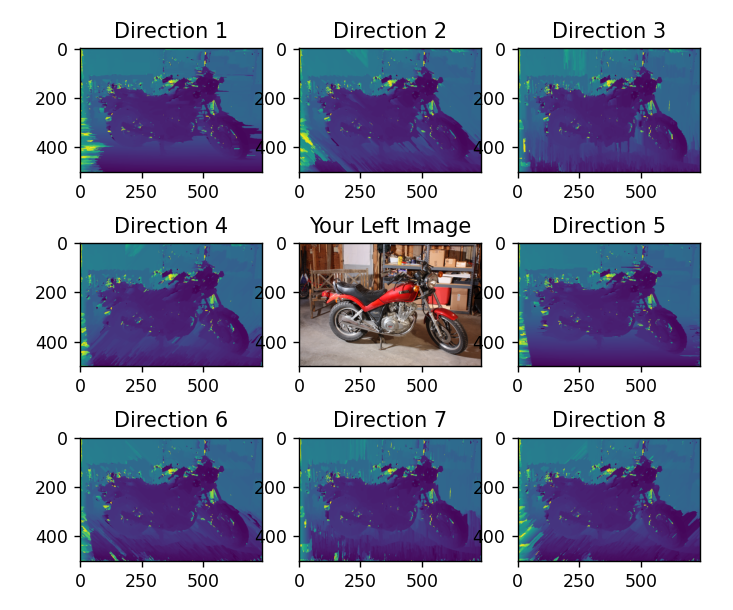
|  |  |
| --- | --- |
| **Left Image** | **Right Image** |
|  |  |

For our own rectified image pair (“my\_image\_left.png” and “my\_image\_right.png”), we ran the full SGM pipeline using the “Your image playground” section. Since our images were approximately 2.5X **larger** than the provided example, we increased the parameters to better match the larger scale and expected disparity range: **WIN\_SIZE = 5** (instead of 3) to use a larger matching window, and **DISPARITY\_RANGE = 60** (instead of 20) to cover the wider horizontal shift between the views. We kept the smoothness penalties the same (**COST1 = 0.5**, **COST2 = 3**) because they control the relative regularization behaviour rather than the image resolution.

**The Result:**



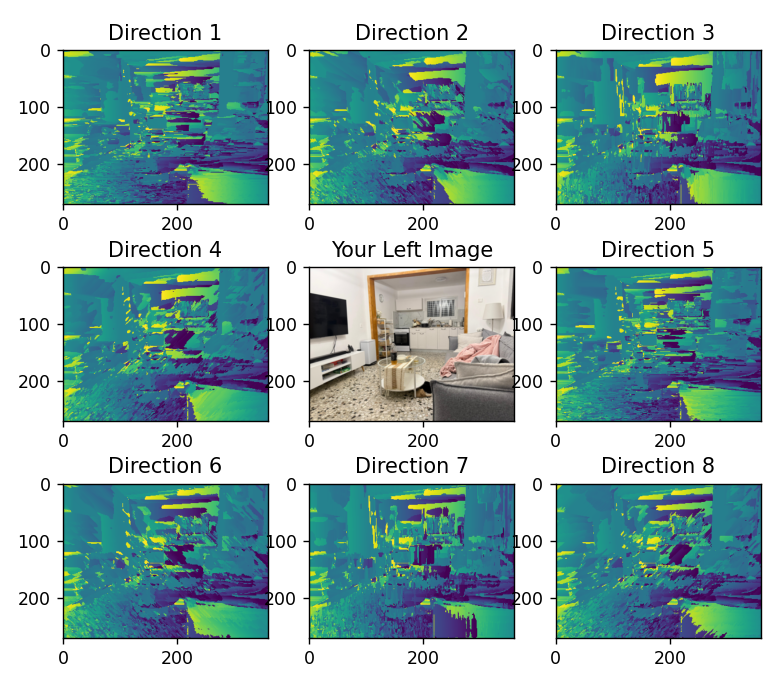
**Per each direction:**



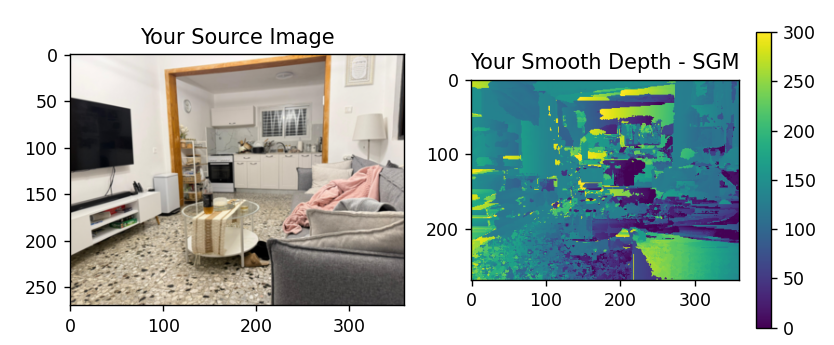
We also tested the method on our own images taken with two phones positioned side by side. **These are the photos we captured for evaluation:**

|  |  |
| --- | --- |
| **Left Image** | **Right Image** |
|  |  |

The pixel differences between the two phone images were quite large, so we set the disparity range to 300. This made the computation slow, so we reduced the resolution of the photos.

**The resulting depth map is shown below:**

**The final result:**



In our results, the ceiling (although physically far) was assigned unexpectedly high disparity values, while the textured floor produced noisy and unstable estimates. This behaviour is likely caused by the two phones not being perfectly aligned and by the fact that the stereo pair was not rectified.

# Bonus Part:

## Part 14

Task: Suggest and implement a new metric to measure distances which replaces SSD. Show naive labelling, Dynamic Programming in a single direction and SGM using the metric you suggested. Compare your results with the SSDD results you obtained in the previous items.

Instead of SSD we use **SAD – Sum of Absolute Differences** over the window:

We preferred SAD because SSD penalizes intensity differences too aggressively. The quadratic term amplifies even small lighting variations, while SAD handles these changes more gently.

|  |  |
| --- | --- |
| **Naive using SSDD** | **Naive using SADD** |
|  |  |

**For each direction:**

|  |  |
| --- | --- |
| **Semi-Global Mapping using SSDD** | **Semi-Global Mapping using SADD** |
|  |  |

**The Final SGM result:**

|  |  |
| --- | --- |
| **Final SGM using SSDD** | **Final SGM using SADD** |
|  |  |

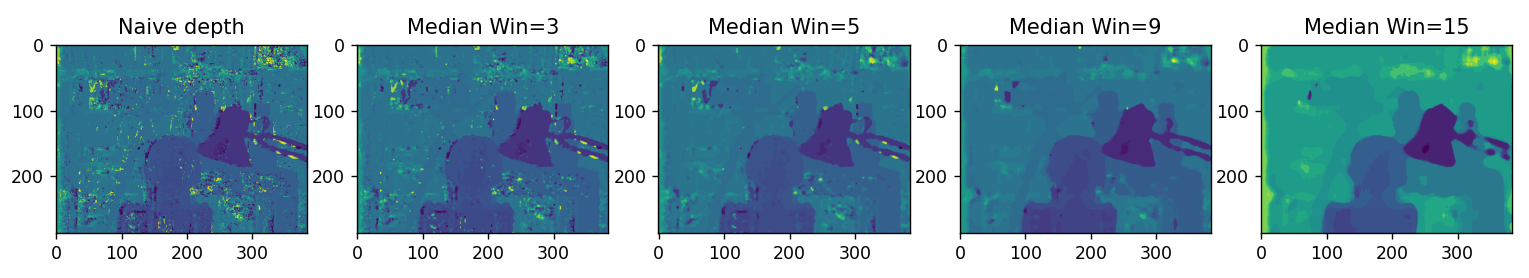
SADD produces a noisier disparity map because its linear penalty creates weaker cost contrast between competing disparities, making it harder for SGM to suppress incorrect matches. However, because SADD does not strongly amplify mismatches, the influence of individual scan directions is reduced, resulting in fewer visible directional streaks. In contrast, SSDD strongly penalizes large differences, leading to smoother regions but also to longer, directionally biased structures. This illustrates a trade-off between smoothness and directional artifacts depending on the chosen cost metric.

## Part 15

Task: Suggest and implement an algorithm to smooth the depth image which replaces the methods suggested in the exercise. You may use SSD as a distance measurement or the metric suggested in 14. Compare your results with the Dynamic Programming and SGM results you obtained in the previous items.

We suggest replacing the smoothing methods used in the exercise with a post-processing median filter applied directly to the depth map. After computing the disparity map using SSD, we smooth it by replacing each pixel with the median disparity value in a local neighbourhood. This approach reduces isolated noise while preserving depth edges. Compared to Dynamic Programming and SGM, the result is visually smoother in homogeneous regions, but it does not enforce global consistency or handle occlusions as effectively as SGM.

**The Result:**



We experimented with different window sizes for the median filter. Smaller windows (3 and 5) reduced some noise but left noticeable speckle artifacts, while a very large window (15) over-smoothed the depth map and blurred object boundaries. The best visual result was obtained with a window size of 9, which provided a good balance between noise reduction and edge preservation.

## Part 16

Task: What are the differences between the results of the depth image from 15 to SGM?

|  |  |
| --- | --- |
| **SGM** | **Median Filter** |
|  |  |

The median-filtered depth map produces a smoother result than the naive labelling and is significantly faster to compute, since it is a simple local post-processing operation. However, it smooths depth values uniformly and may blur fine structures and object boundaries. In contrast, SGM aggregates costs along multiple directions, allowing it to preserve edges and capture coherent object shapes more accurately. While SGM is computationally more expensive, it produces sharper depth discontinuities and more geometrically consistent results than median smoothing.