GTU Department of Computer Engineering

CSE 463 - Fall 2024

Homework 3 Report

Emirkan Burak Yılmaz
1901042659

1 Contents

1 Contents 2
2 Introduction to Oversegmentation 3
2.1 SLIC (Simple Linear Iterative Clustering) 3
2.2 Felzenszwalb's Efficient Graph-Based Segmentation 3
2.3 Quickshift 3
2.4 Watershed 3
3 Proposed Stereo Correspondence Algorithm 4
3.1 Oversegmentation 4
3.2 Histogram Creation for the Segments 5
3.3 Segment Matching 5
3.4 Creating Disparity Map 6
3.5 Calculating Error Metrics with Ground Truth 7
4 Results 8
4.1 SLIC 8
4.2 Felzenszwalb 10
5 Conclusion 11

2 Introduction to Oversegmentation

Oversegmentation algorithms in image processing divide an image into multiple small, coherent regions or segments, known as superpixels. These segments are more perceptually meaningful than individual pixels and typically respect object boundaries better than a grid-based representation. The goal of oversegmentation is not to achieve a high-level segmentation that distinguishes different objects, but rather to simplify and reduce the complexity of an image into more manageable pieces. This approach is often used as a preprocessing step for other image analysis tasks such as segmentation, tracking, and object recognition. Here's a brief overview of several popular oversegmentation algorithms

2.1 SLIC (Simple Linear Iterative Clustering)

SLIC is an efficient superpixel algorithm that clusters pixels in the combined five-dimensional color and image plane space to form compact, nearly uniform superpixels. It starts with an initial grid of cluster centers and iteratively refines their positions based on pixel similarity and spatial proximity, balancing color similarity and spatial proximity through a compactness parameter.

2.2 Felzenszwalb's Efficient Graph-Based Segmentation

Felzenszwalb's algorithm segments images by representing the image as a graph, where pixels are nodes and edges represent the similarity between neighboring pixels. It then finds a minimum spanning tree and iteratively merges regions based on a defined criterion involving edge weights and region sizes. It is efficient, handles large variations in segment sizes, and captures perceptually important boundaries.

2.3 Quickshift

Quickshift is a mode-seeking algorithm that segments images based on density peaks in the joint appearance and spatial domain. It links each pixel to the nearest neighbor that increases the density estimate until convergence, forming clusters around density peaks. It is non-parametric, preserves boundaries well, and does not require a predefined number of segments.

2.4 Watershed

The watershed algorithm segments images based on the concept of topographic surface analysis. It treats the gradient magnitude of an image as a topographic surface and finds watershed lines where water would flow between basins. Markers are used to indicate the initial locations of the basins. It is intuitive and effective for segmenting objects with high contrast boundaries.

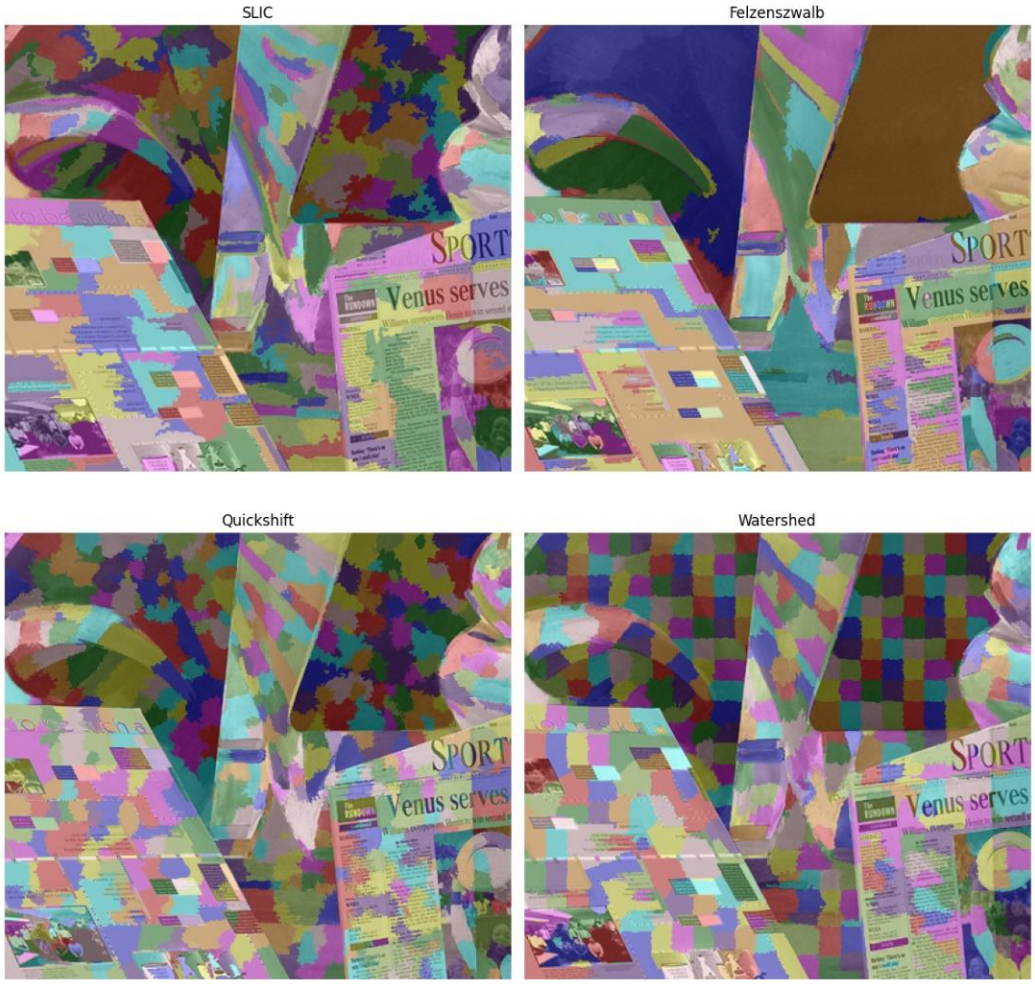


Figure 1: Different oversegmentation methods applied on Venus image from Middlebury dataset.

3 Proposed Stereo Correspondence Algorithm

The proposed stereo correspondence algorithm encompasses several key steps aimed at accurately matching image segments between rectified stereo pairs. Initially, the stereo images underwent preprocessing to enhance features and reduce noise. Oversegmentation methods were then applied to partition the images into superpixels, serving as the basis for segment matching. Subsequently, histograms representing the color and texture features of the segments were computed for each pair of corresponding epipolar lines. Matching proceeded by comparing these histograms along the epipolar lines and selecting the best matches based on a similarity metric. Finally, disparity values were assigned to the matched segments, resulting in a disparity map that represents the depth information of the scene. Below are the detailed descriptions of each step:

3.1 Oversegmentation

Oversegmentation methods are employed to divide the images into superpixels or smaller regions. This step aims to group pixels with similar attributes, such as color or texture, together while maintaining boundaries between distinct objects or regions in the scene. Common oversegmentation algorithms include SLIC, Felzenszwalb, Quickshift, and Watershed.

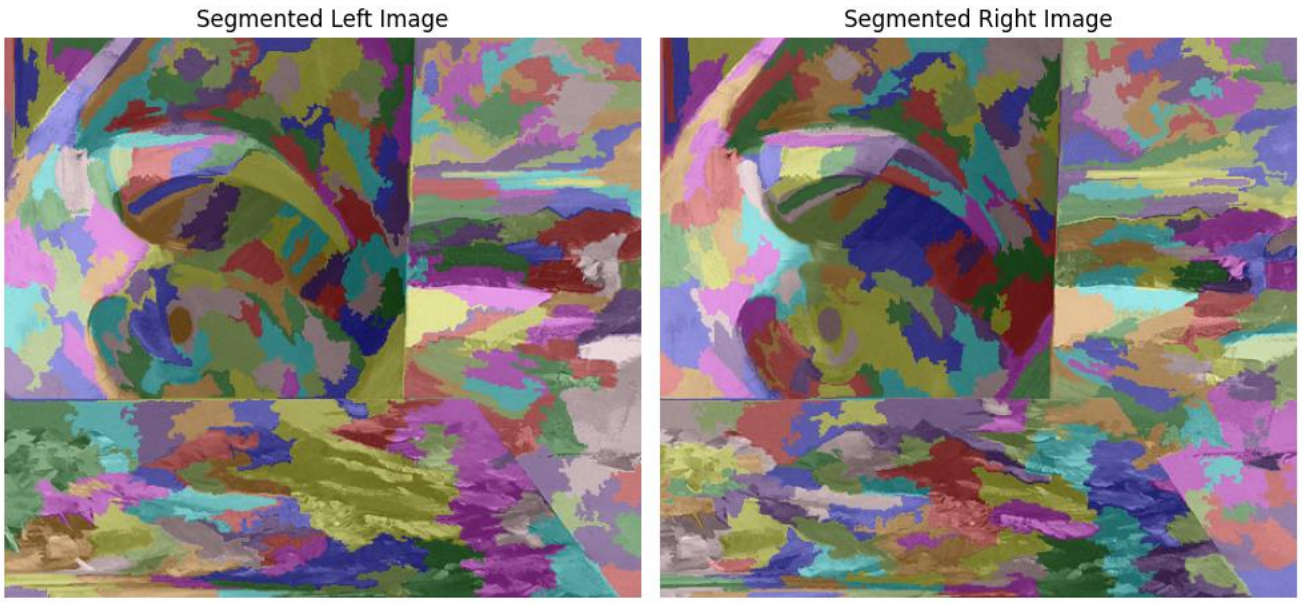


Figure 2: Oversegmentation with SLIC algorithm

3.2 Histogram Creation for the Segments

For each pair of corresponding epipolar lines, histograms representing the color and texture features of the segments are computed. These histograms serve as descriptors for the segments and capture important information about their characteristics, allowing for effective comparison and matching.

```
def compute_segment_histograms(image, segments):
    histograms = {}
    for segment_id in np.unique(segments):
        mask = segments == segment_id
        hist = cv2.calcHist([image], [0, 1, 2], mask.astype(np.uint8), [8, 8, 8], [0, 256, 0, 256, 0, 256])
        hist = cv2.normalize(hist, hist).flatten()
        histograms[segment_id] = hist
    return histograms
```

Figure 3: Computing histogram for the given segment

3.3 Segment Matching

In conventional stereo matching methods, the disparity for a pixel is usually determined by scanning a window across the disparity range, selecting the target pixel based on minimizing the error. However, in this algorithm, computed segments are utilized instead. Matching of the segments between the stereo images is achieved by comparing the histograms along the epipolar lines. The similarity metric Bhattacharyya distance is employed to measure the similarity between histograms and identify the best matches. This approach is notably more efficient compared to traditional window-based techniques.

```python
def match_segments(left_segments, right_segments, left_histograms, right_histograms, disparity_range=32):
    height, width = left_segments.shape

    matches = {}  # Segments matches from left to right
    scores = {}  # Scores for the best matches
    disparity_map = np.zeros(left_segments.shape)

    for row in range(height):
        prev_li = -1
        for col in range(disparity_range, width):
            li = left_segments[row, col]

            best_offset = 0
            best_score = float('inf')
            for offset in range(disparity_range):
                # Compare the corresponding segments
                ri = right_segments[row, col - offset]
                left_hist = left_histograms[li]
                right_hist = right_histograms[ri]
                score = cv2.compareHist(left_hist, right_hist, cv2.HISTCMP_BHATTACHARYYA)

                if score < best_score:
                    best_score = score
                    best_offset = offset
                    if best_score < scores.get(li, float('inf')):
                        scores[li] = score
                        matches[li] = ri

            disparity_map[row, col] = best_offset

    return matches, disparity_map
```

Figure 4: Segment matching and disparity calculation

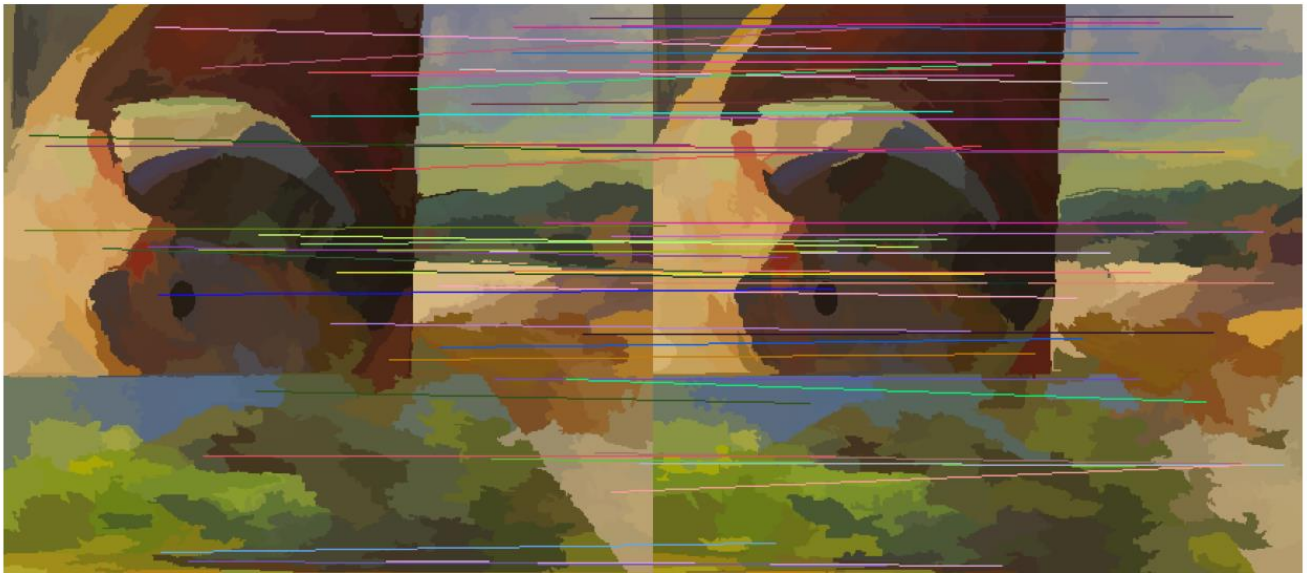


Figure 5: Matched segments

3.4 Creating Disparity Map

Once the matches are found, a disparity map is generated by calculating the disparity between matched segments. The computed disparity values represent the depth information of the scene, indicating the difference in position between corresponding points in the stereo images.

```python
# Update the disparity map by assigning the average disparity value to each segment
def average_disparity_map(disp_map, segments):
    average_disp_map = np.zeros_like(disp_map)
    unique_segments = np.unique(segments)
    for segment_id in unique_segments:
        mask = segments == segment_id
        average_disp = np.mean(disp_map[mask])
        average_disp_map[mask] = average_disp
    return average_disp_map
```

Figure 6: Averaging disparity values inside the segment

Figure 7: Calculated disparity map

3.5 Calculating Error Metrics with Ground Truth

Finally, error metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are computed to evaluate the accuracy of the disparity map generated by the algorithm compared to the ground truth disparity map provided by the Middlebury dataset.

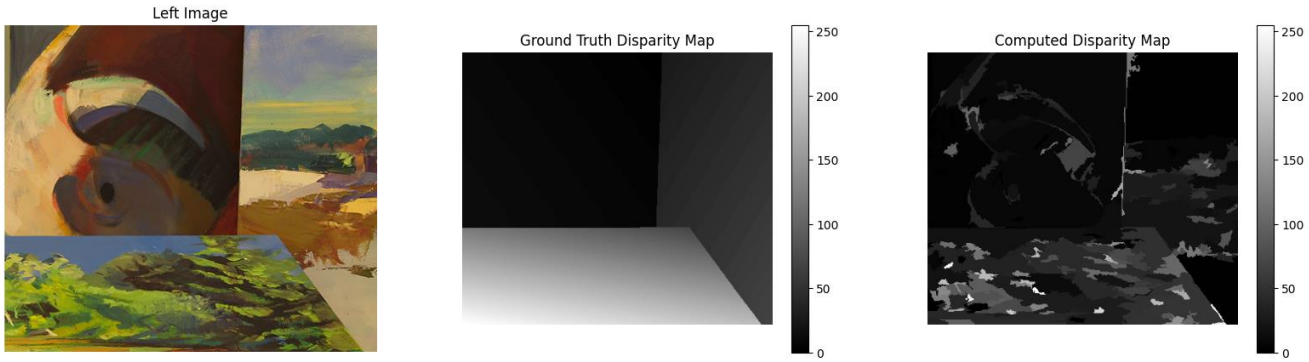


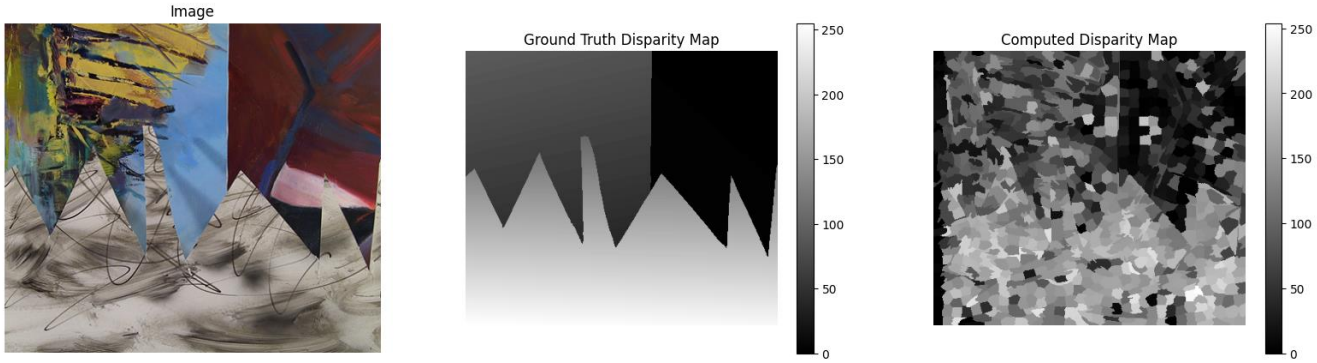
Figure 8: Comparison with Middlebury ground truth

4 Results

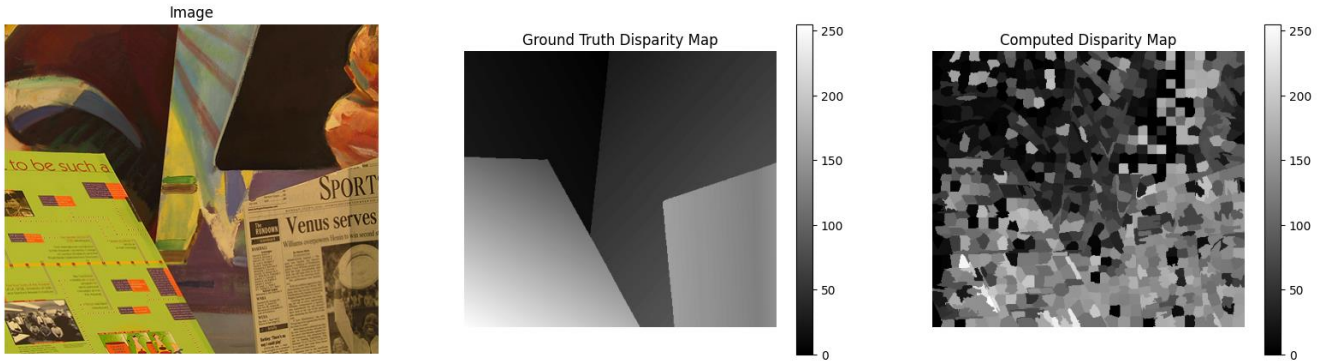
The report presented experimental results for four oversegmentation algorithms, specifically focusing on SLIC and Felzenszwalb. For a comprehensive view of all the results, please refer to the “Tests & Results” part of the accompanying Jupyter notebook.

4.1 SLIC

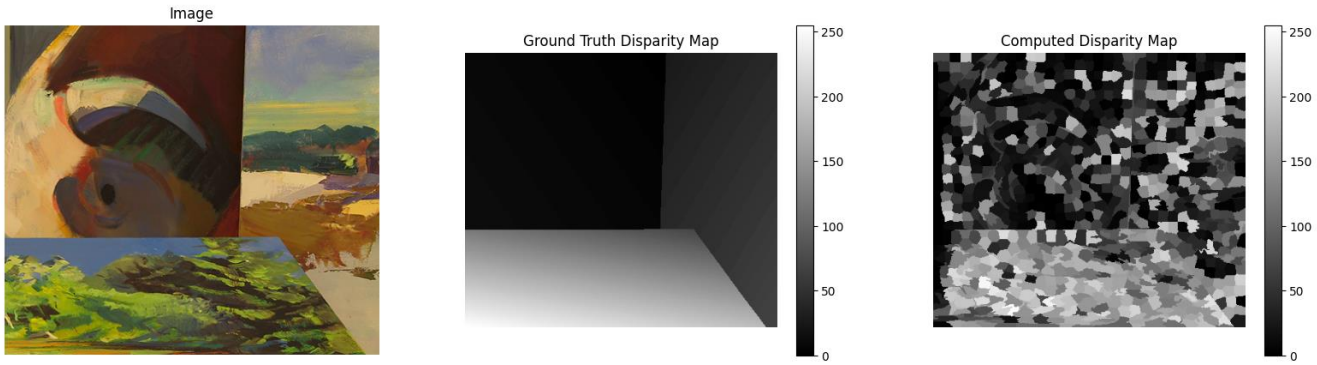
```
Category: sawtooth, MAE: 75.71, RMSE: 84.02
```



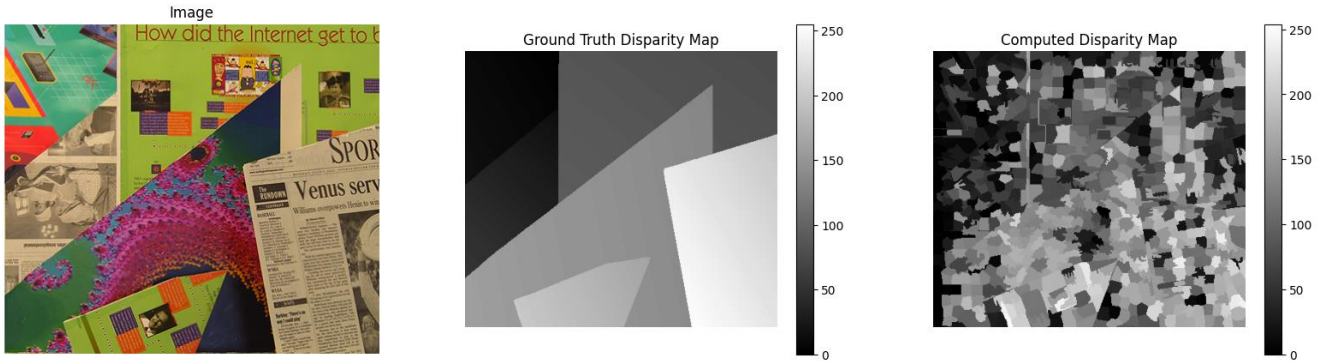

```
Category: venus, MAE: 67.19, RMSE: 74.25
```



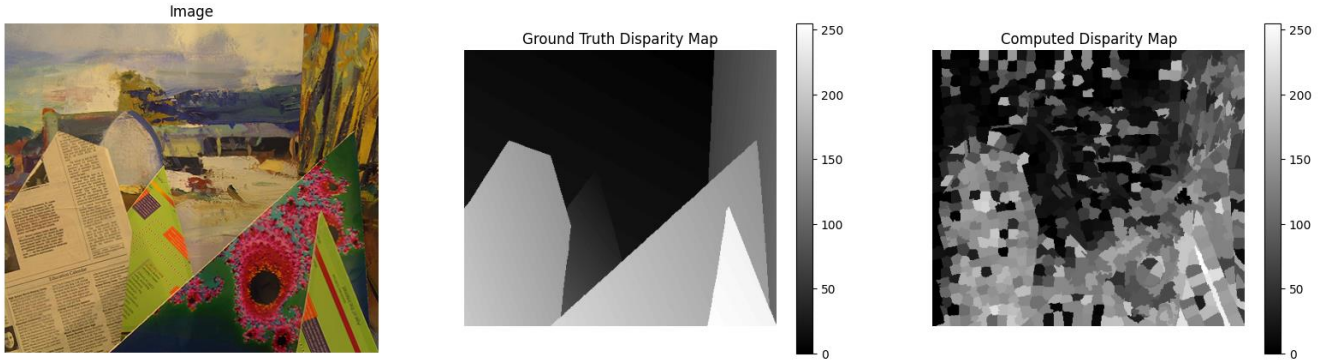

```
Category: bull, MAE: 56.90, RMSE: 65.58
```



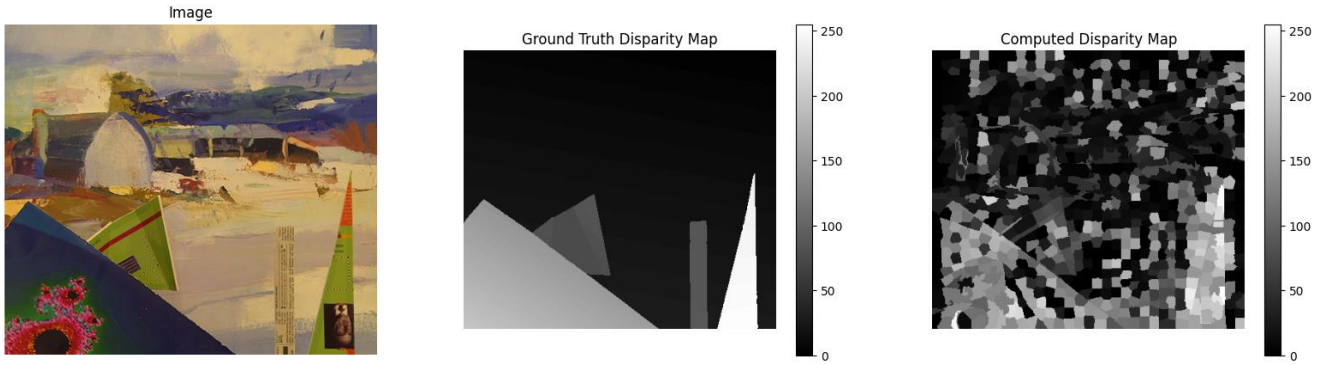
Category: poster, MAE: 88.08, RMSE: 95.46



Category: barn1, MAE: 62.26, RMSE: 69.61

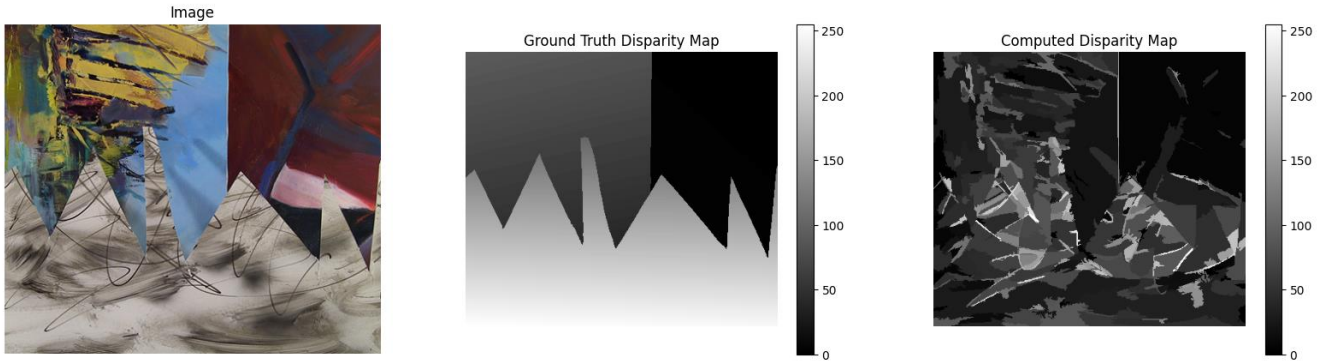


Category: barn2, MAE: 44.86, RMSE: 52.35

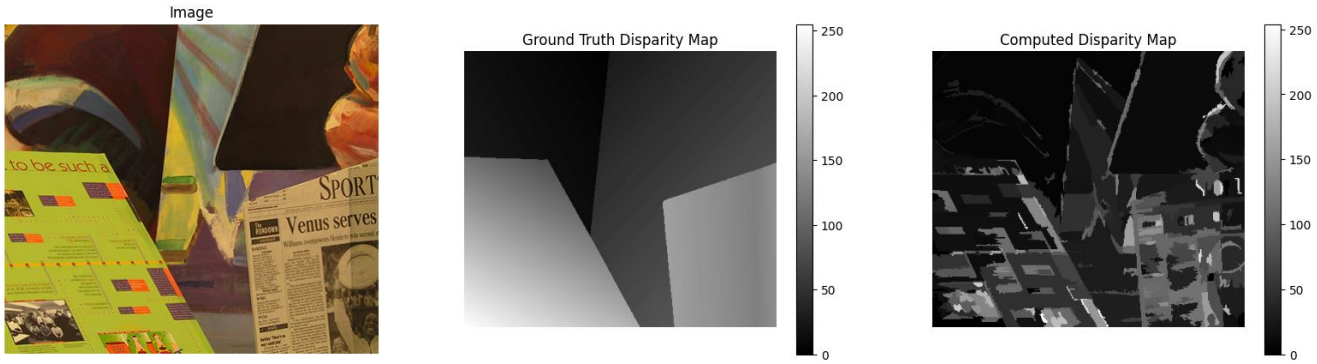


4.2 Felzenszwalb

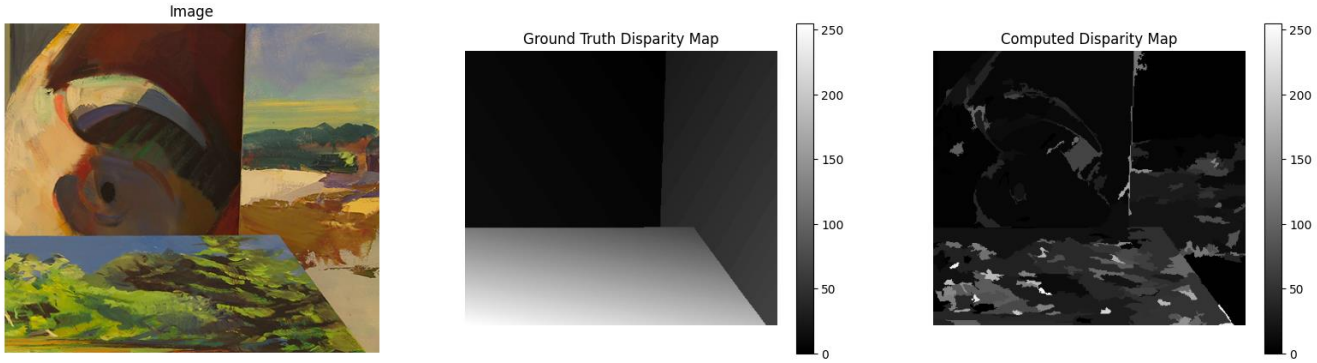
Category: sawtooth, MAE: 78.09, RMSE: 86.60



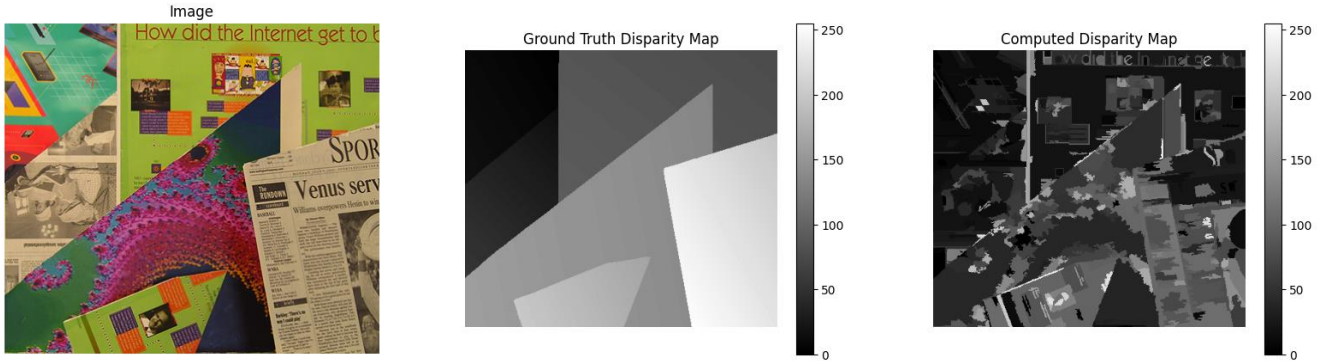
Category: venus, MAE: 69.19, RMSE: 76.14



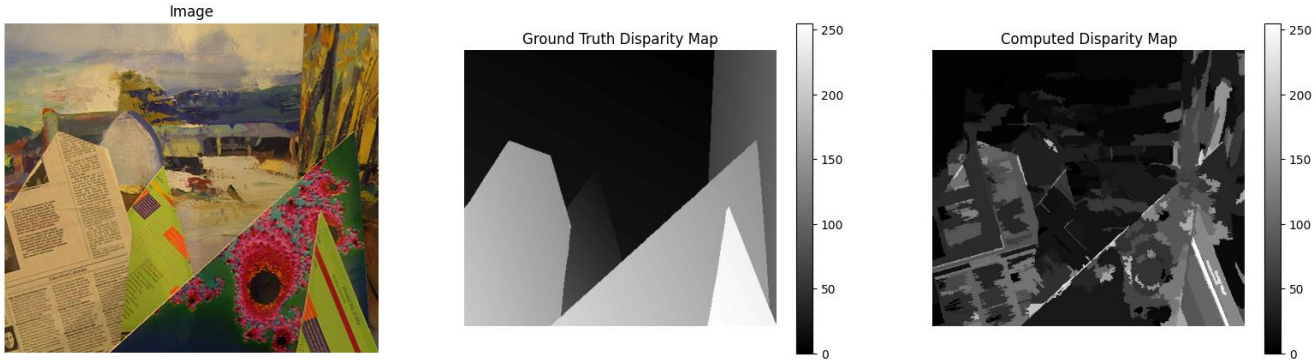
Category: bull, MAE: 59.17, RMSE: 67.87



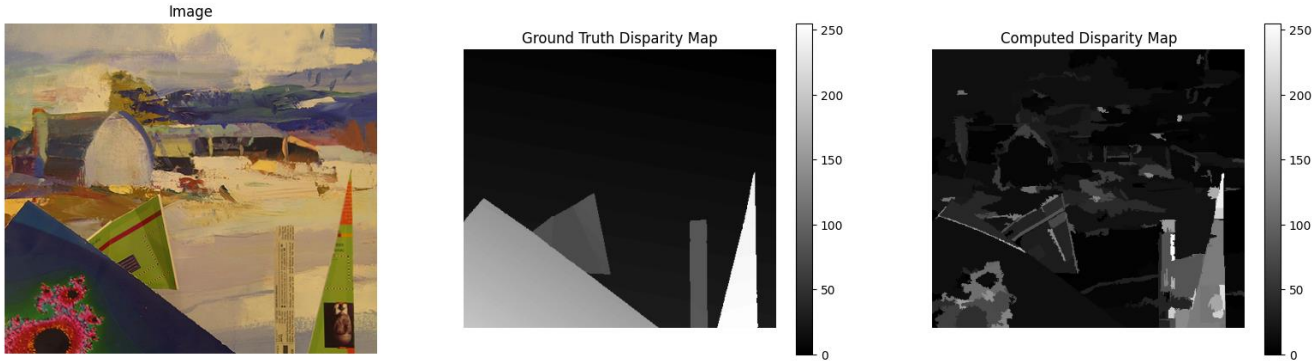
Category: poster, MAE: 89.77, RMSE: 97.14



```
Category: barn1, MAE: 63.62, RMSE: 71.01
```




```
Category: barn2, MAE: 46.28, RMSE: 53.77
```



5 Conclusion

The performance of the algorithm was evaluated using both qualitative and quantitative measures. Qualitatively, the disparity maps generated by the algorithm exhibited some noise but preserved object boundaries, providing a general view of the depth. Quantitative analysis revealed disparities between the algorithm's results and the ground truth data from the Middlebury dataset. Mean Absolute Error (MAE) and Mean Squared Error (MSE) metrics were utilized to quantify the accuracy of disparity estimation. While the algorithm achieved competitive results on certain scenes such as "Venus" and "Bull," it encountered difficulties with scenes containing textureless regions or occlusions, resulting in higher error rates. Overall, the proposed algorithm shows promise for stereo correspondence tasks but requires further refinement to address its limitations and improve its performance in challenging scenarios.