CS 532 - 3D Computer Vision
Optional End of Course Project

Akshay Atam[1]

1. Introduction

The goal of this project is to implement Multi View Patch Match algorithm for depthmap generation. Originally introduced by Barnes et al.[1], PatchMatch employs a randomized search strategy to match local image patches across multiple views. It harnesses the concept of "good patchiness," where patches that exhibit similar appearance and structure are likely to correspond to the same underlying 3D structure in the scene.

All that follows on this task's background I gained from Stevens Institute of Technology's CS 532 - 3D Computer Vision course guided by Professor Enrique Dunn. This project falls under the domain of Multi View Stereo (MVS), where the primary goal of MVS is to estimate the geometry and appearance of a scene from multiple images taken from different viewpoints. It enables us to create accurate, dense, and photorealistic 3D models, capturing intricate details and surface properties.

2. Dataset Description

For this project, a subset of Stretcha MVS Dataset was used namely,

- Entry-P10
- Fountain-P11
- Herz-Jesu-P8

The dataset contains 10, 11, and 8 images respectively. A sample of three images from each dataset was chosen to implement the Patch Match algorithm and the resulting depthmap was to be shown at each iteration.

3. Code overview

1. It imports the necessary libraries, including numpy, cv2 (OpenCV), os, and tqdm. It defines several functions:
    - `get_values(path, filename)`: Reads camera calibration parameters (K, R, t) from a file and returns them as numpy arrays.

[1]Stevens Institute of Technology, Hoboken, NJ 07030

- `get_P(path, filename)`: Reads projection matrix (P) from a file and returns it as a numpy array.
- `SAD(ref1, ref2)`: Computes the Sum of Absolute Differences between two image patches.
- `SSD(ref1, ref2)`: Computes the Sum of Squared Differences between two image patches.
- `NCC(ref1, ref2)`: Computes the Normalized Cross-Correlation between two image patches.
- `to_3d(pos, z, K, R, T)`: Projects a 2D point to 3D space using camera calibration parameters.
- `to_2d(pos, P)`: Projects a 3D point to 2D image coordinates using a projection matrix.
- `project(loc, depth, K0, R0, T0, P1)`: Projects a 2D point with depth to another view using camera calibration and projection matrices.
- `photoconsistency_measure(original_pos, ref_image, other_img, positions, window_size)`: Evaluates the photoconsistency between corresponding patches in the reference and other images.

2. The reference image (`img_ref`) and its camera calibration parameters (K_{ref}, R_{ref}, t_{ref}) and projection matrix (P_{ref}) is read along with the the other images (`img1`, `img2`) and their respective camera calibration parameters and projection matrices.
3. The lower and upper limits for the random depth map generation is set and a random depth map is initialized with the shape of the reference image.
4. Scan directions ('RL', 'LR', 'UD', 'DU') tells us the scan direction and a window size tells us the neighborhood size.
5. A number of patch match iterations is set and it starts a loop over the number of iterations.
6. Within each iteration, it loops over the rows and columns of the depth map. It then selects a scan direction based on the current iteration and finds the neighbor pixel based on the scan direction.
7. A random value within the specified depth range is set. It projects the current pixel, neighbor pixel, and random pixel into the other image (`img1`) using the projection matrices.
8. It evaluates the photoconsistency between the corresponding patches in the reference and other images and selects the best depth hypothesis based on the photoconsistency evaluation.
9. The algorithm then updates the depth map based on the best depth hypothesis and saves the depth map after each iteration.

4. Observation

It can be observed that the resulting depthmap does not take the shape of the respective structure it should represent after 10 iterations. However, we can see that at the 10^{th} iteration, there is some depth observed which indicates that more iterations are needed for the algorithm to converge. It is also noted that the scale of image is also responsible for the resulting accuracy of the image. Both affect the accuracy in different ways.

1. **Number of Iterations:** The PatchMatch algorithm is an iterative algorithm that improves the depth map estimation over multiple iterations. Increasing the number of iterations allows the algorithm to refine the depth map further and potentially improve the accuracy of the depth estimation. However, a higher number of iterations also increases the computational time required for the algorithm. It is a trade-off between accuracy and computational efficiency.
2. **Scaling Factor:** A smaller scaling factor restricts the search space, focusing the search for matching patches within a smaller region around each pixel. This can lead to faster convergence of the algorithm but may also result in a more limited search range and potentially less accurate depth estimation. On the other hand, a larger scaling factor allows for a broader search range but may require more iterations for convergence.

5. Conclusion and Future Scope

The project was to implement the Patch Match algorithm for depthmap generation. Although, the choice of hyperparameters was not adequate, due to time constraints the choices were made. However, there is a lot of scope for improvement. The code needs to be computationally fast in every scaling factor and a relationship between scaling factor, window size, and iteration size must also be kept in mind.

6. Output

Parameters:

- scaling factor = 0.25
- number of iterations = 10

The output is shown at the end of the report.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, D. B. Goldman, Patchmatch: A randomized correspondence algorithm for structural image editing, ACM Trans. Graph. 28 (3) (2009) 24.

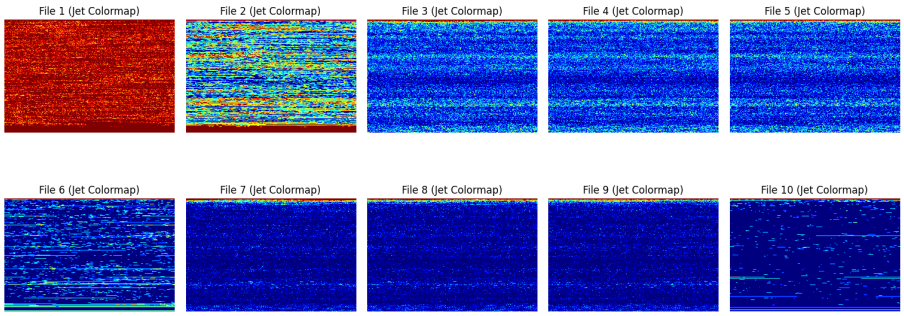

Figure 1: Jet colormap

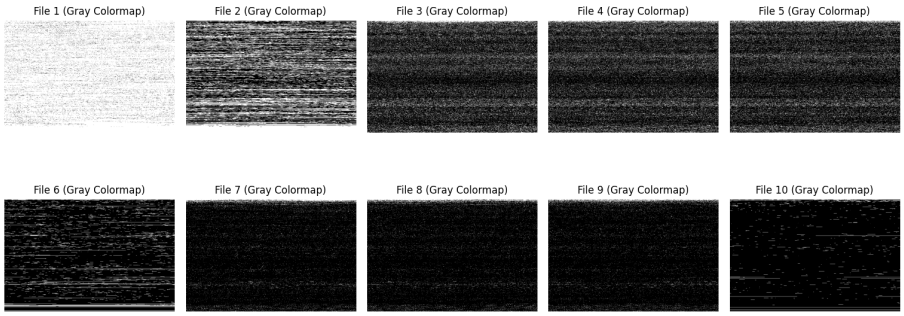

Figure 2: Gray colormap

Figure 3: Entry-P10 SAD

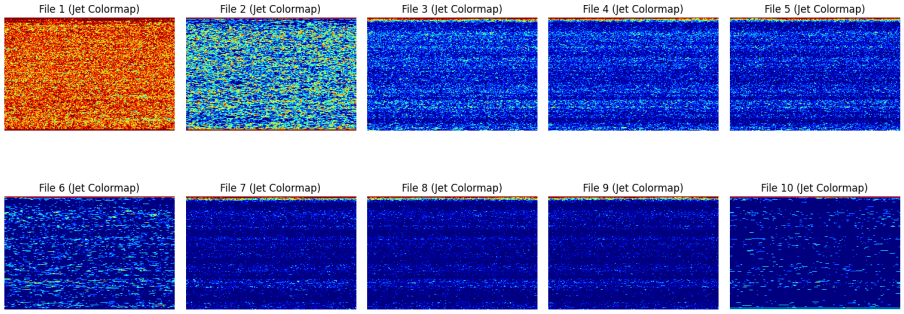

Figure 4: Jet colormap

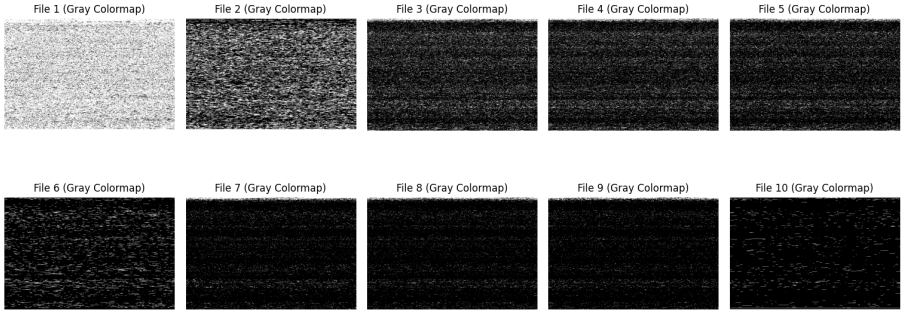

Figure 5: Gray colormap

Figure 6: Entry-P10 SSD

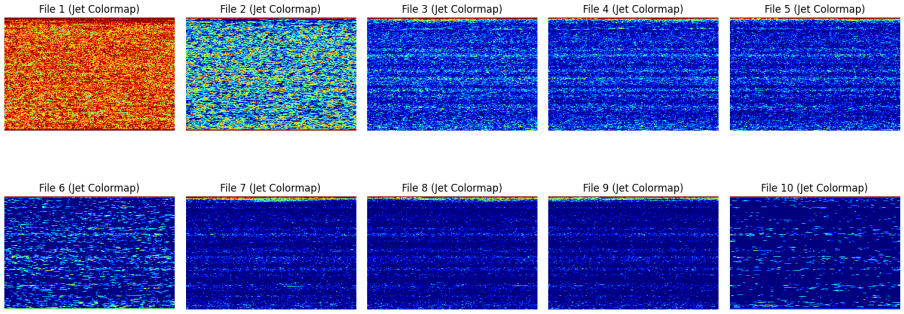

Figure 7: Jet colormap

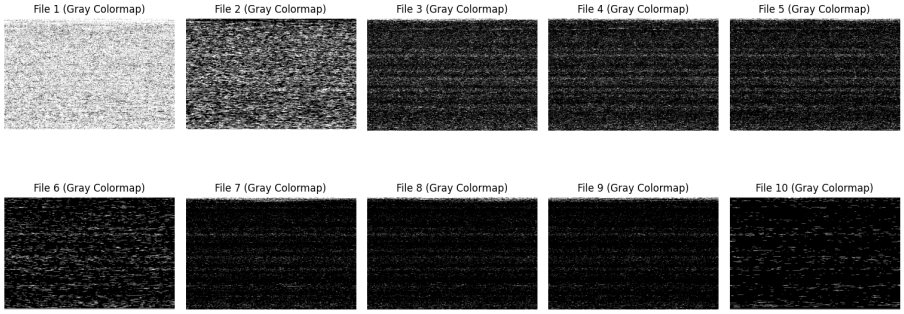

Figure 8: Gray colormap

Figure 9: Entry-P10 NCC

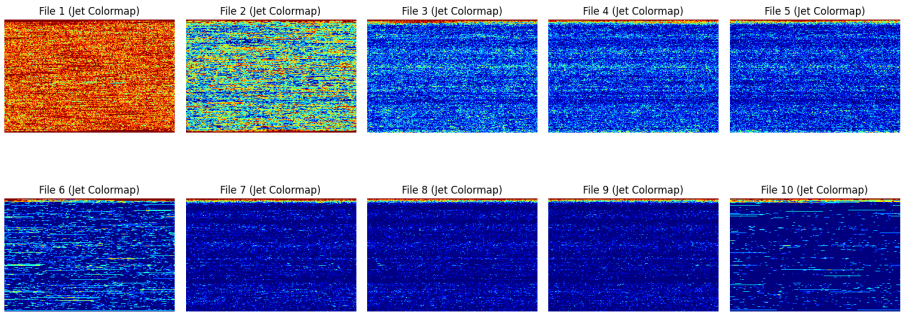

Figure 10: Jet colormap

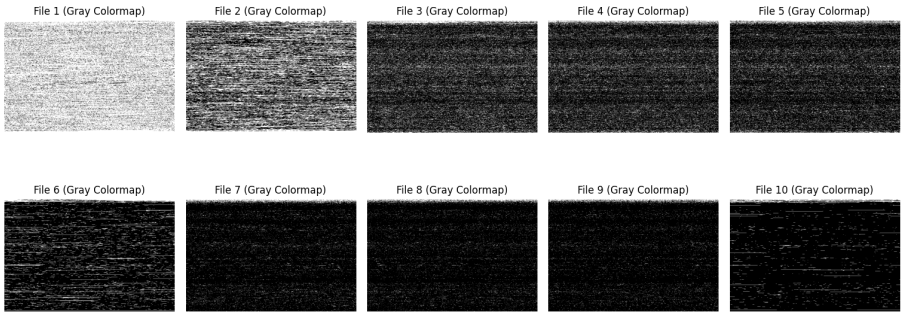

Figure 11: Gray colormap

Figure 12: Fountain-P11 SAD

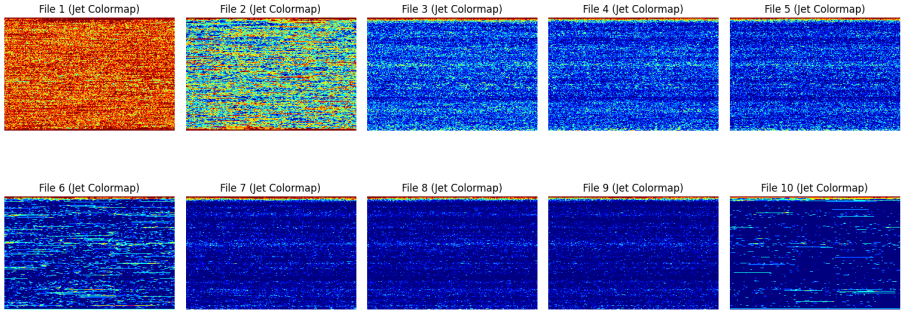

Figure 13: Jet colormap

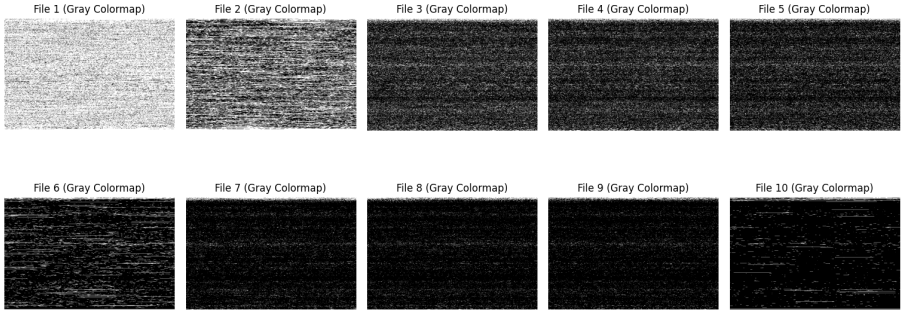

Figure 14: Gray colormap

Figure 15: Fountain-P11 SSD

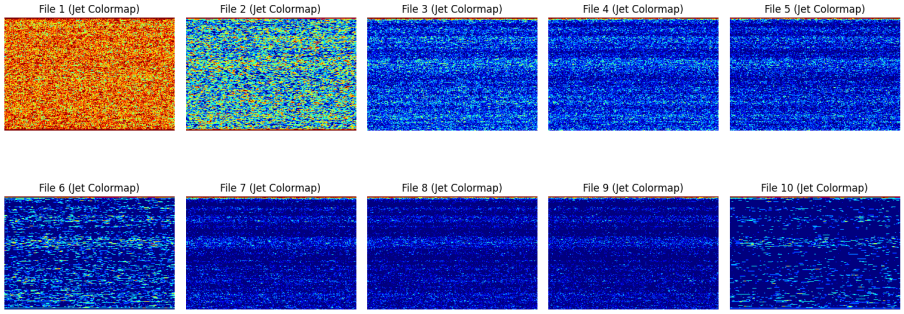

Figure 16: Jet colormap

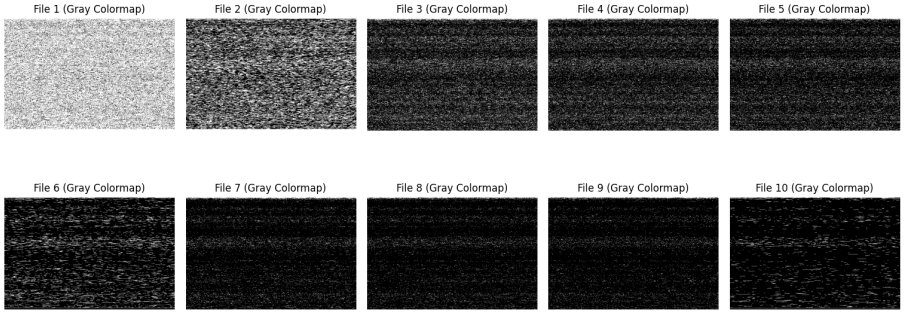

Figure 17: Gray colormap

Figure 18: Fountain-P11 NCC

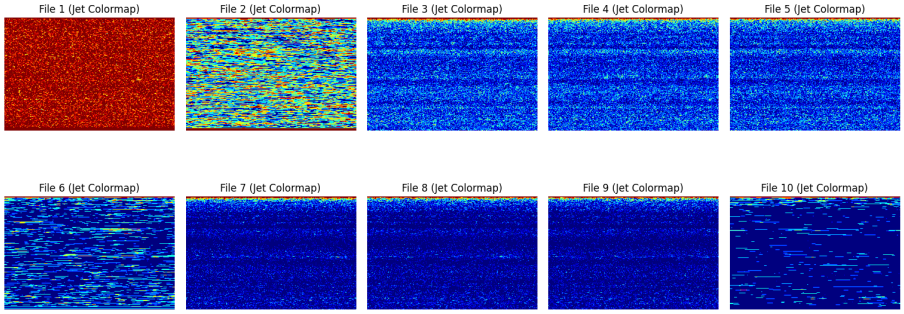

Figure 19: Jet colormap

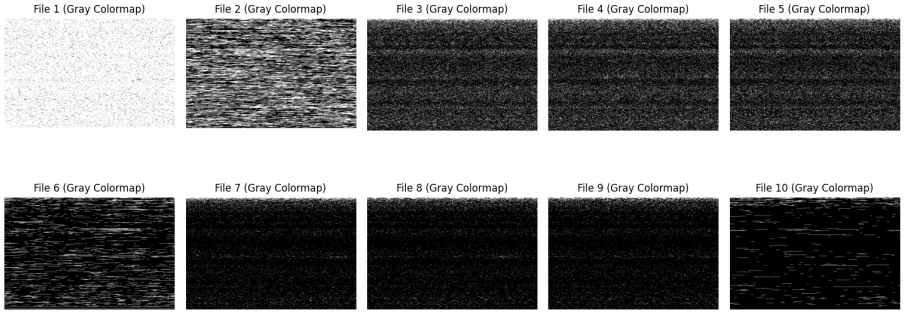

Figure 20: Gray colormap

Figure 21: Herz-Jesu-P8 SAD

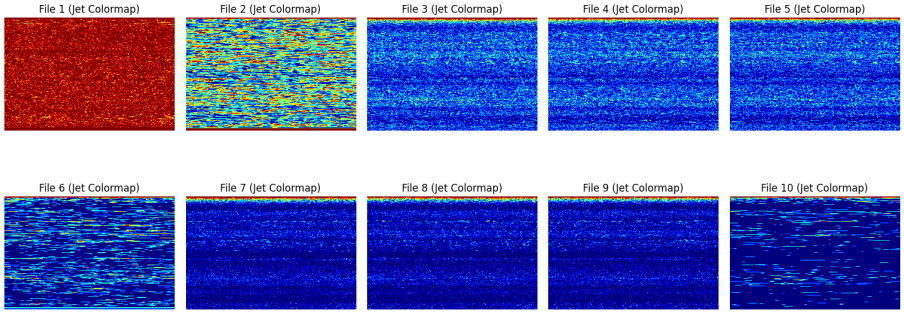

Figure 22: Jet colormap

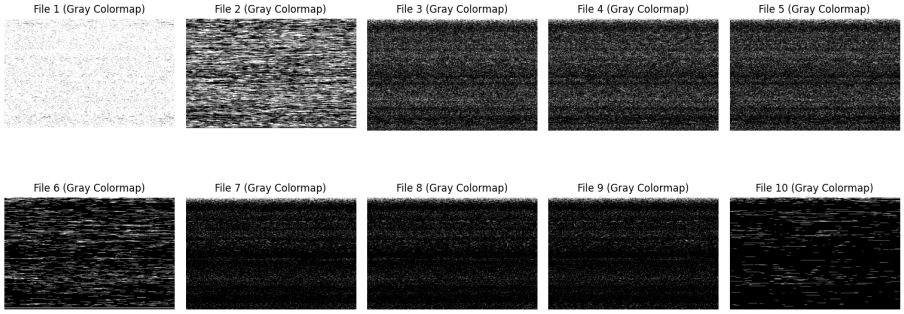

Figure 23: Gray colormap

Figure 24: Herz-Jesu-P8 SSD

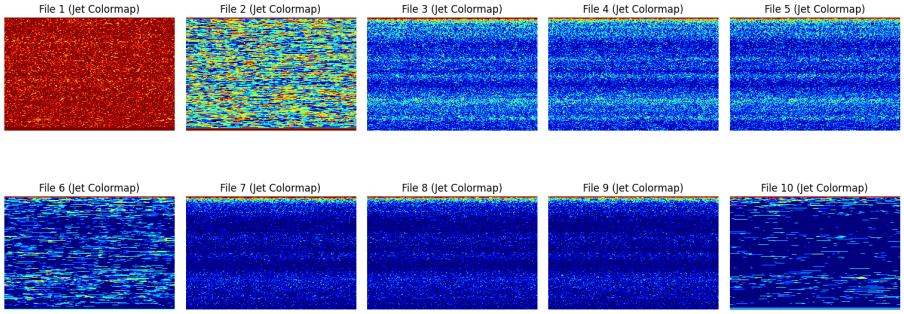

Figure 25: Jet colormap

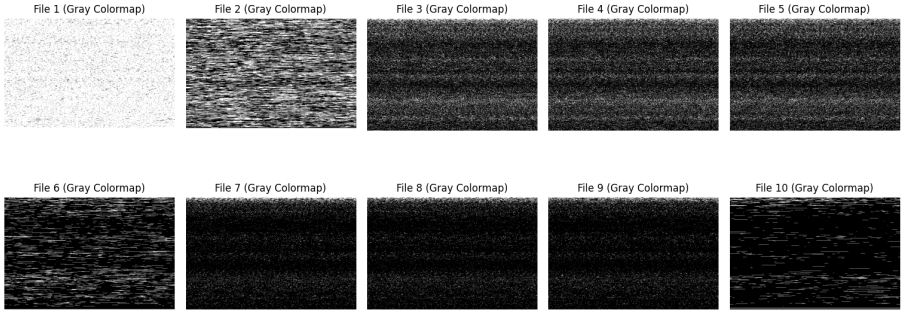

Figure 26: Gray colormap

Figure 27: Herz-Jesu-P8 NCC