INF573 - Linear View Interpolation with Image Rectification and Disparity Mapping

Chen Jiabin, Zheng Léon

December 2018

Abstract

Our project goal was to understand and implement Seitz and Dyer's **view interpolation algorithm** [3] and try to use it on our own unrectified pictures to evaluate the quality of the interpolation. This interpolation method consists of three steps: **rectification**, **linear interpolation** using disparity mapping, and **derectification**. Our results show the validity of this method, but there are two main issues: computing a precise disparity mapping between the rectified images; and computing a valid rectification of the images while knowing the transformation matrices used in the process so that we can compute derectification. Finally, we achieved clean interpolation of intermediate rectified views by using our own implementation of **Kolmogorov and Zabih's graph cuts stereo matching algorithm** [4].

1 Introduction

In artificial intelligence technologies, DeepMind researchers [1] have recently work on **scene representation**, which is the process of converting visual sensory data into concise descriptions. It can be used for example to create new views of a specific subject starting from a number of pictures taken from given point of views. To learn more about this field, we decided to work in this project more specifically on view interpolation.

View interpolation aims to **generate**, from pictures of a same scene taken from different points of a trajectory in space or in time, **a new view** of this same scene from any points of this trajectory. In this project, we will focus on guessing the intermediate view of a scene situated on a straight line that connects **two reference views** of this same scene. View interpolation, as it was introduced in the 1990s, was originally used for image rendering, and today, it can be applied for example to generate new views in Street View, when we want to see a street scene from an angle that hasn't been taken in picture.

Several techniques exist for view interpolation. The first approach, like in Chen and William's paper in 1993 [2], is to reconstruct the 3D scene from the reference views, so that we can project the reconstruction on the intermediate

view. Here, we use another approach, used by Seitz and Dyer in their 1995's paper [3], to interpolate a view **without using 3D reconstruction**. This method consists in three steps: **rectification**, **linear interpolation**, **derectification**.

Our project's goal is to use this algorithm to interpolate views with our own pictures, and observe the quality of the interpolation that we can achieve. In this report, we will analyze this algorithm and its limits when we try to implement it in practice, so that we can find how to improve the interpolation results. We will see how we can achieve better results by implementing a **graph cut disparity mapping algorithm** for improving the linear interpolation step. At the end of this report, we will discuss about the quality of our interpolation results.

2 Linear view interpolation algorithm

2.1 Algorithm overview

This algorithm is detailed in Seitz and Dyer's 1995 paper [3]. The two reference views of the scene will be called 'left' and 'right' images. The figure 1 is a summary of this algorithm.

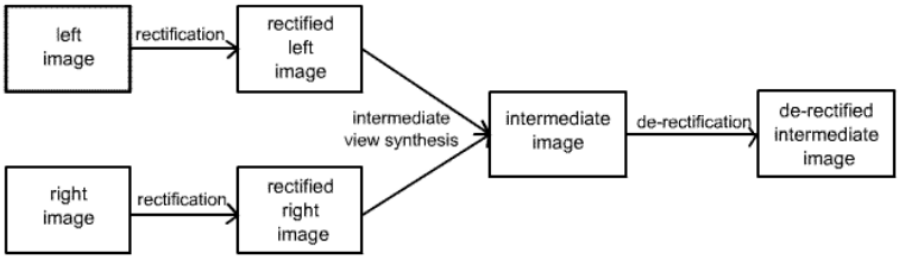

Figure 1: View interpolation algorithm, which consists in three main steps: rectification of the left and right images, synthesis of the intermediate view, and derectification of the generated view.

2.1.1 Rectification step

The *rectification* step is based on **keypoints matching** between the left image, named I_1, and the right image, I_2.

Assuming that we have obtained a list of n matched keypoints, $(r_k^1, r_k^2)_{k=1,\dots,n}$, where $r_k^i = (x_k^i, y_k^i)$ are the coordinates in image I_i $(i = 1, 2)$ of the keypoints.

We choose, for each image I_i, the centroid of all the keypoints r_k^i as the origin of the image, such that $\sum_{k=1}^{n} r_k^i = [0, 0]^T$. Then, let T_i be the coordinates of the top-left corner of I_i in this reference frame.

Using this reference frame for the coordinates (x_k^i, y_k^i), we define the measurement matrix M as:

$$M = \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 \\ y_1^1 & y_2^1 & \cdots & y_n^1 \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ y_1^2 & y_2^2 & \cdots & y_n^2 \end{pmatrix} \tag{1}$$

Let U, Σ, V be the matrices of the **single value decomposition** of matrix M, such that:

$$\boxed{M = U\Sigma V} \tag{2}$$

Let $U^{'}$ be the matrix formed by the first 3 columns of U. We then define the matrices U_1 and U_2 such that:

$$U^{'} = \begin{pmatrix} U_1 \\ U_2 \end{pmatrix}$$

The direction of epipolar lines in I_1 and I_2 can be determined from U_1 and U_2 as follows: partition $U_i = [A_i|d_i]$ where $A_i : 2 \times 2, d_i : 2 \times 1$. We then define:

$$B_i = \begin{pmatrix} A_i^{-1} & -A_i^{-1}d_i \\ 0 & 1 \end{pmatrix}$$

$$U_1^{'} = U_1 B_2$$

$$U_2^{'} = U_2 B_1$$

Let $(x_i, y_i)^T$ be the third column of $U_i^{'}$. The epipolar lines in image I_i make an angle $\theta_i = arctan(y_i/x_i)$ with the horizontal axis. We can then define the rotation matrix

$$R_{-\theta_i} = \begin{pmatrix} cos\theta_i & sin\theta_i \\ -sin\theta_i & cos\theta_i \end{pmatrix}$$

. Then, we have

$$R_{-\theta_2} U_2^{'} B^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & s & 0 \end{pmatrix}$$

and we define

$$H_s = \begin{pmatrix} 1 & 0 \\ 0 & 1/s \end{pmatrix}$$

Finally, we obtain the *rectified* images $I_1^{'}$ and $I_2^{'}$ using the following equations:

$$\boxed{I_1^{'} = R_{-\theta_1}(I_1 + T_1)} \tag{3}$$

$$\boxed{I_2^{'} = H_s R_{-\theta_2}(I_2 + T_2)} \tag{4}$$

2.1.2 Interpolation step

The article [3] shows that interpolation of rectified monotonic images always produces valid in-between views of a scene. After rectification, the epipolar lines are horizontal, and, assuming we have a disparity mapping points of the two rectified images, we can linearly interpolate the view as follow.

Assume p_1 and p_2 are corresponding points in rectified images $I_1^{'}$ and $I_2^{'}$ respectively. Then the corresponding point p_i in the **rectified intermediate image** $I_i^{'}$ with a chosen $1 \leq i \leq 2$ is:

$$\boxed{p_i = (2-i)p_1 + (i-1)p_2} \tag{5}$$

with corresponding intensity

$$\boxed{I_i^{'} = (2-i)I_1^{'} + (i-1)I_2^{'}} \tag{6}$$

2.1.3 Derectification step

After this interpolation step, we obtain the rectified intermediate image that we need to *derectify* to obtain the final interpolated view of the scene.

To do that, given the $1 \leq i \leq 2$ used to interpolate the rectified view in the previous section, we define:

$$\theta_i = (2-i)\theta_1 + (i-1)\theta_2$$

$$s_i = (2-i) + (i-1)s$$

$$T_i = (2-i)T_1 + (i-1)T_2$$

We finally **derectify** the rectified interpolated view $I_i^{'}$ using:

$$\boxed{I_i = R_{\theta_i} H_{1/s_i} I_i^{'} - T_i} \tag{7}$$

2.2 Algorithm implementation

Our implementation of the algorithm is written in C++, in the directory *view-interpolation*.

2.2.1 Keypoints matching

The first step of the algorithm is to match keypoints that will be use in the measurement matrix M in equation (1). The quality of the rectification only depends of the quality of the matching.

In our implementation, we firstly use **AKAZE in OpenCV** to detect keypoints in each image. Then, we do a brute force matching of keypoints, and only keep the N matchings with the smallest distance in a certain norm between the descriptors of the two keypoints. N is a parameter that we can tune

depending on the input images, but it is typically between 100 and 300 for a good rectification.

Finally, we use the OpenCV's implementation of **RANSAC** to eliminate the outlier matchings. The typical keypoints matching that we can obtain is shown in figure 2.

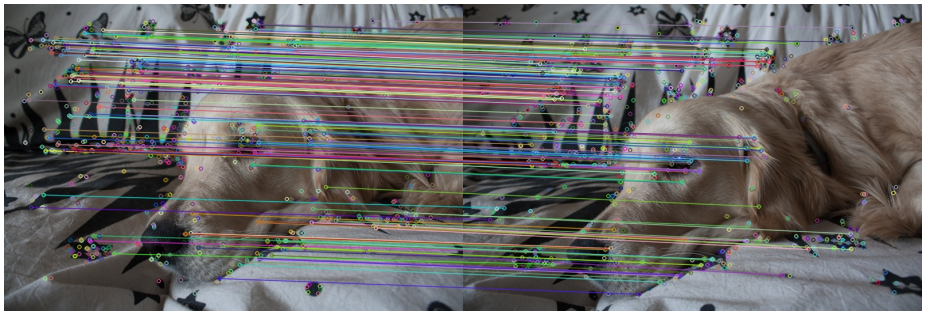

Figure 2: Keypoints matching obtained after AKAZE brute force matching, selection of the matchings with the smallest distance between descriptors, and elimination of outlier matchings by RANSAC.

2.2.2 Single value decomposition

The single value decomposition implementation used to compute equation (2) is the one from **Eigen library of C++**.

2.2.3 Disparity mapping

In a first approach, we use OpenCV's implementation of **StereoSGBM algorithm** to compute the disparity mapping between the two rectified images. There are a lot of parameters to tune for the disparity computation (minDisparity, numDisparities, blockSize, P1, P2, disp12MaxDiff, uniquenessRatio, speckleWindowSize, speckleRange). For each input images, if we fine tune those parameters, we can compute a good disparity mapping, well enough to make a clean view interpolation, just as the figure 3 shows.

2.3 First results, limits and possible improvements

2.3.1 Validity of the interpolation method

Assuming we have a perfect rectification of two images, and a known disparity mapping between the two rectified images, can we interpolated correctly the the intermediate view given the equation (5) and (6)? The answer is yes, just as the result in figure 4 shows. This result **validates the linear interpolation method** using equations (5) and (6).

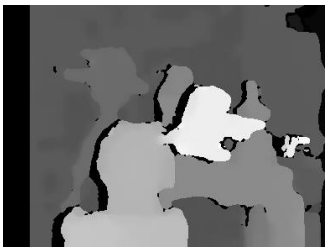

Figure 3: Disparity mapping computed with OpenCV's StereoSGBM class with specific parameters. The quality of the computation is high enough so that it can be used to cleanly interpolate the intermadiate view.

Figure 4: The top picture is the view from the left, the bottom picture is the view from the right. The middle picture is the interpolated view. The left and right images are already rectified, and the disparity mapping used is the one from the figure 3. We can observe how the algorithm has generated the intermediate view by observing the new position of the statue or the lamp in the middle picture.

2.3.2 Results of our implementation on unrectified pictures

To evaluate the performance of our implementation of this view interpolation algorithm, we tested it on two pictures of a pen. The results are shown in figure 5.

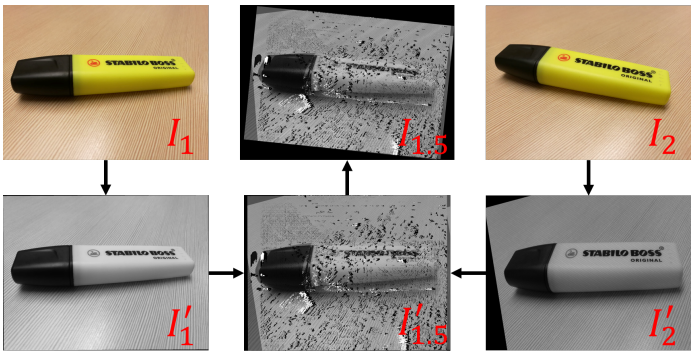

Figure 5: Unrectified images are named I_i, while rectified images are named $I_i^{'}$. The final interpolated view is $I_{1.5}$. Interpolated views are not clean because of a bad disparity mapping. This is mainly caused by a unvalid rectification where corresponding points in $I_1^{'}$ and $I_2^{'}$ are not at the same horizontal level. However, the orientation of the pen in $I_{1.5}$ is interesting, because we can feel that this picture represents an intermediate view between I_1 and I_2

We can observe that the **rectification step isn't valid**, because corresponding points in $I_1^{'}$ and $I_2^{'}$ are not at the same horizontal level. Thus, the disparity mapping cannot be done correctly, and as a consequence, the rectified interpolated view $I_{1.5}^{'}$ isn't clean. The derectified interpolated view $I_{1.5}$ obtained with equation (7) is still interesting, because we can observe that the orientation of the pen in this intermediate view is between the orientation of the pen in the two reference views.

Therefore, with this result, we can identify *two main difficulties* for the view interpolation problem:

- first, we need to construct a **valid rectification** of the input images, so that all the corresponding points are on the same horizontal axis;
- second, we need to obtain a **precise disparity mapping**, because the interpolation is based on the equations (5) and (6).

2.3.3 Difficulty of rectification

By using the rectification method of the equations (3) and (4), we don't always obtain rectification in which corresponding points on the rectified left and right images are on the *same horizontal axis*, which is problematic for the computation of disparity mapping.

It seems that the precision of the rectification depends on the **position of the keypoints** that are kept as valid matching for the measurement matrix M in equation (1). In the figure 6, we see that corresponding points close to the keypoints that have been selected in the rectification process have the same horizontal level, whereas corresponding points that are far from those keypoints are not at the same horizontal level.

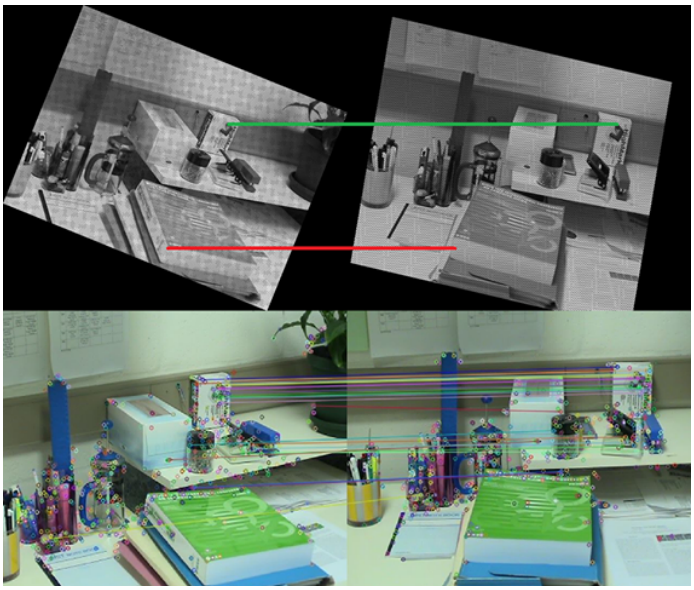

Figure 6: Top images: rectified images obtained with equations (3) and (4). Bottom images: matched keypoints used in the measurement matrix M in equation (1). We observe that most of the selected keypoints are situated on the small box near the wall, rather than on the green book. As a result, on the rectified images, corresponding points of the small box have the same horizontal level (green line), whereas corresponding points of the book are not on the same horizontal level (red line).

In order to get a valid rectification of the whole picture, we need to choose matched keypoints that are **well distributed** over all the objects in the picture. This is difficult specially when there are some texture-less parts in the picture.

One can use other rectification methods, by using for example OpenCV's implementation of rectification algorithm. But the problem of not using Seitz and Dyer's rectification method is that the rotation matrix R_{θ_i}, the scale matrix H_{1/s_i} and the translation matrix T_i used to derectify the rectified interpolated view with equation (7) are unknown. That means, if we rectify images with another method without using equation (3) and (4), we can still interpolate the rectified intermediate view, but we cannot derectify it.

2.3.4 Difficulty of disparity mapping

The main difficulty of disparity mapping is to obtain a proper results without having to **fine tune** the parameters of the StereoSGBM object in OpenCV for each input images. That is why we implemented a graph cut disparity mapping so that we can have a better disparity computation.

3 Graph cuts stereo matching algorithm

The *graph cut stereo matching algorithm* that we chose is the one developped by Kolmogorov and Zabih, and presented in the paper [4]. The main advantage of this algorithm is that we don't need to fine tune a lot of parameters like what we did with OpenCV's StereoSGBM object, and **results are better**. However, the computation time is **longer**.

The algorithm is implemented in Python, in the directory *graph-cuts-stereo-matching*.

3.1 Algorithm overview

3.1.1 Definitions

Let $\mathcal{L}$ (resp. $\mathcal{R}$) be the set of pixels of the left image I_1 (resp. right image I_2). Then p (a pair of coordinates) denotes a pixel location from the left image and q a pixel from the right image. We also assume that the disparity lies in the $I_{disp} = [x_{min}, x_{max}]$ since the images are supposed to be in rectified epipolar geometry.

Let $\mathcal{A} \subset \mathcal{L} \times \mathcal{R}$, called the set of *assignments*, be the set of pairs of pixels which may potentially correspond. Then, for every assignment $a = (p, q) \in \mathcal{A}$, we define the disparity $d(a) = q - p$. Let us say that two assignments $a_1 = (p_1, q_1)$ and $a_2 = (p_2, q_2)$ are neighbors, written $a_1 \sim a_2$, if p_1 and p_2 are adjacent and $d(a_1) = d(a_2)$.

A *configuration* is a map $f : \mathcal{A} \to \{0, 1\}$. When $f(a) = 1$, the assignment a is *active*, that means it is p and q corresponds. In the opposite, when $f(a) = 0$, a is *inactive*. A configuration is *unique* if for all pixels p (resp. q), there is at most one active assignment involving p (resp. q). If every assignment $a = (p, q)$ involving p is inactive then p is not matched with any pixel. In that case, p is called occluded under the configuration f.

3.1.2 Algorithm principle

The Kolmogorov and Zabih's is based on **assignments**. The goal is to find a unique configuration f^* that **minimizes** a certain *energy function*. We can then access to the disparity mapping by looking at all the active assignments.

3.1.3 Energy function

For any configuration f, we define the following energy function:

$$\boxed{E(f) = E_{data}(f) + E_{occlusion}(f) + E_{smoothness}(f) + E_{uniqueness}(f)} \quad (8)$$

Data term When an assignment $a = (p, q)$ is active, the pixels p and q should be similar, that means, in a certain metric D, $D(a)$ is small. For a gray scale image, $D(a)$ can be for example the difference of intensity between

pixels p and q. Thus, the data term is defined as:

$$E_{data}(f) = \sum_{a, f(a)=1} D(a) \tag{9}$$

Occlusion term A good configuration f should maximize the number of matches, that means, minimize the number of occluded pixels. Since the number of occluded pixels is an affine function of the number of inactive assignments, the penalty for inactive assignment is K:

$$E_{occlusion}(f) = \sum_{a, f(a)=0} K \tag{10}$$

Smoothness term Two assignments a_1 and a_2 should be both active or both inactive if they are neighbors. Otherwise, there is a penalty of V_{a_1,a_2}:

$$E_{smoothness}(f) = \sum_{a_1 \sim a_2} V_{a_1,a_2} \cdot \mathbb{1}_{\{f(a_1) \neq f(a_2)\}} \tag{11}$$

Uniqueness term We enforce with this term the uniqueness of the configuration:

$$E_{uniqueness}(f) = \sum_{\substack{a_1=(p,q_1) \\ a_2=(p,q_2) \\ q_1 \neq q_2}} \infty \cdot \mathbb{1}_{\{f(a_1)=f(a_2)=1\}} + \sum_{\substack{a_1=(p_1,q) \\ a_2=(p_2,q1) \\ p_1 \neq p_2}} \infty \cdot \mathbb{1}_{\{f(a_1)=f(a_2)=1\}} \tag{12}$$

3.1.4 Expansion move

Unfortunately, the energy (8) cannot be represented by a graph on which we can apply graph cuts. Therefore, we minimize the energy by iterative *expansion moves*.

For a given configuration f and $\alpha \in I_{disp}$ disparity value, we say that a configuration f' is an α-*expansion move* of f if:

$$f(a) = 1 \text{ and } d(a) = \alpha \implies f'(a) = 1 \tag{13}$$

$$f(a) = 1 \text{ and } d(a) \neq \alpha \implies f'(a) = 0 \tag{14}$$

In terms of assignments,

- any active assignment with disparity α remains active;
- any inactive assignment with disparity different from α remains inactive;
- any other assignment can change state (active/inactive).

In order to minimize the energy (8), given a configuration f and a disparity α, we choose the α-expansion move $f^{'}$ that decreases the most the energy $E(f^{'})$. To do that, for any expansion move $f^{'}$, we introduce a function g_α, defined as:

$$g_\alpha(a) = \mathbb{1}_{\{f(a) \neq f'(a)\}} \tag{15}$$

We also introduce a **graph representable energy** $E_{f,\alpha}$ which verifies, for any g_α associated to an α-expansion move $f^{'}$:

$$\boxed{E_{f,\alpha}(g_\alpha) = E(f^{'})} \tag{16}$$

Therefore, we can represent this energy with a **graph** $\mathcal{G}$ with a source and a sink, where the vertices represent the assignments, and where the edges have specific weights, so that the cost of a *s-t cut*, which corresponds to a certain function g_α, is equal to the energy $E_{f,\alpha}(g_\alpha)$. We then apply a graph cuts algorithm to determine the *minimum cut*, so that we can find the g_α that minimizes the energy $E_{f,\alpha}(g_\alpha)$. This function g_α is then used to determine the α-expansion move $f^{'}$ that decreases the most the energy $E(f^{'})$.

Thus, the algorithm to find the configuration f that minimizes the energy $E(f)$ is the following one:

Data: Set of assignments $\mathcal{A}$, disparity range I_{disp}
Result: Configuration f that minimizes energy (8)
choose an initial configuration f;
for $i \leftarrow 1$ **to** N *(number of expansion moves)* **do**
  $\alpha \leftarrow$ random value in I_{disp};
  construct the graph $\mathcal{G}$ representing $E_{f,\alpha}$ defined in (16);
  apply graph cuts to find minimum cut;
  $g_\alpha \leftarrow$ function defined in (15) corresponding to the minimum cut;
  $f^{'} \leftarrow$ expansion move corresponding to g_α;
  $f \leftarrow f^{'}$;
end

Algorithm 1: Expansion move algorithm to minimize energy (8)

3.2 Algorithm implementation

By implementing this algorithm with the indications given in the article [4], we obtain **acceptable** disparity mappings just as we can see in figure 7. We can easily detect the relief of the scene through the progression of the disparity values. But there are still a lot of **occluded pixels** for which we cannot compute a disparity value because they represent parts of the background that are occluded by the foreground objects from one image to the other one.

The main problem of our implementation is the **time complexity**. The step that takes the most time is the construction of graph $\mathcal{G}$ and maxflow algorithm. In fact, this implementation consist of a series α-expansion, in which we need to build the graph, apply maxflow and then update disparity. In each

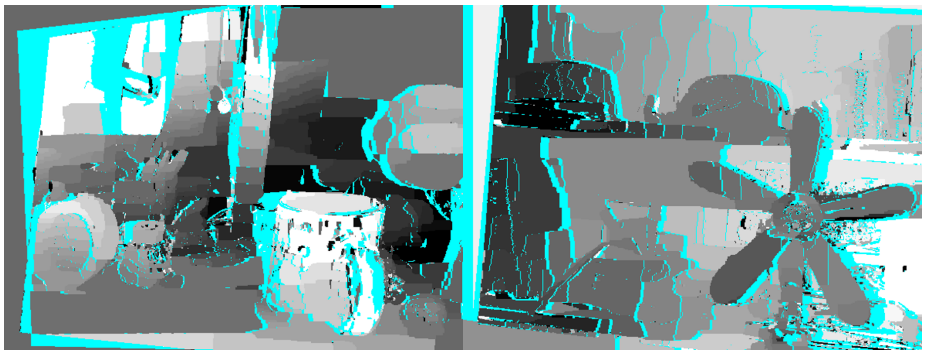

Figure 7: Disparity mapping of our own rectified pictures computed with Kolmogorov and Zabih's stereo matching algorithm. Occluded pixels are represented in blue. The result of the interpolated view computed with this mapping can be seen in figure 8.

iteration, because we can use only $\mathcal{O}(1)$ to calculate the weights to be added for each assignment, so we take $\mathcal{O}(n)$ to build graph, which n represents the number of pixels in the image. Then if we use Edmond-Karp (finding augmenting paths with breadth-first search) for Maxflow, the complexity is $\mathcal{O}(n^2E)$, which E represents the number of edges. Finally, we need $\mathcal{O}(n)$ to update the disparity matrix.

3.3 Graph cuts stereo matching for view interpolation

To evaluate the performance of implementation of the graph cuts stereo matching algorithm applied to view interpolation, we will not use the rectification step using equations (3) and (4), because this rectification method isn't valid when we use our own unrectified pictures, like what we can see in the figure 6. Instead, we will use **other rectifications methods**, like the one in OpenCV, so that we can have good rectifications, on which we can compute the disparity mapping with our implementation of stereo matching algorithm, and use it to interpolate the rectified intermediate view. We will **not derectify** this generated view.

In figure 8, we represent the results of interpolated rectified views using Kolmogorov and Zabih's graph cuts stereo matching algorithm, on our own pictures, without fine tuning parameters like in OpenCV's StereoSGBM for every inputs.

The result is **satisfying** specially when there are few occluded pixels. When there are too much of them, the generated intermediate view can have some *holes*. To solve this issue, one can develop techniques to fill these holes by guessing their original colors.

Figure 8: Top to bottom: left view, intermediate view, right view. Intermediate views generated with linear interpolation using disparity mapping computed with Kolmogorov and Zabih's graph cuts stereo matching algorithm. Pictures are taken by ourselves. The computed disparity mappings are displayed in figure 7.

4 Conclusion

This project showed the **validity** of the view interpolation method proposed by Seitz and Dyer in 1995 [3], which consists of three steps: **rectification, linear interpolation using disparity mapping, and derectification**. As a result, pictures of our interpolated views are satisfying, just as we can see in figures 4, 5 and 8.

While we managed to compute a relatively precise disparity mapping using Kolmogorov and Zabih's **graph cuts stereo matching algorithm** presented in [4] to improve the quality of intermediate rectified views, we still need to find a solution to **rectify** correctly the left and right images while knowing the **rotation, scale and translation matrices** used in the rectification process so that we can **derectify** the generated intermediate rectified views.

The challenge is also to have good rectification methods and disparity computation in the case when the two initial reference views aren't not as close as on they are on figure 4 or 8, but more **separate** like on figure 5. It is a harder case with more interesting applications.

References

[1] Danilo Jimenez Rezende S. M. Ali Eslamim. Neural scene representation and rendering. *Science*, 2018.

[2] Lance Williams Shenchang Eric Chen. View interpolation for image synthesis. *Apple Company, Inc.*, 1993.

[3] Charles R. Dyer Steven M. Seitz. Physically-valid view synthesis by image interpolation. *University of Wisconsin*, 1995.

[4] Pauline Tan Vladimir Kolmogorov, Pascal Monasse. Kolmogorov and zabih's graph cuts stereo matching algorithm. *Image Processing On Line*, 2014.