Recovery of 3D Urban Scenes:
Reconstruction from Two Images with Known Internal Parameters

Álvaro Budria, Alex Carrillo, Sergi Masip and Adrià Molina

Universitat Pompeu Fabra

February 2022

1 Introduction

In this document, we present our methodology, implementation, and results from different 3D reconstruction approaches. More precisely, we triangulate by matching correspondences between two views and applying the Direct Linear Transform (DLT). Then, we extract camera matrices from these images, given the Essential matrix relating the camera views. We evaluate this method with a geometric reprojection error. After that, we apply triangulation to create a 3D reconstruction from two views.

In the second part, we focus on estimating depth maps using local methods and implementing bilateral weights on these maps. Lastly, we review two approaches for depth-fusion-based 3D reconstruction, one of them classic, the other learning-based.

2 Triangulation with the DLT method

In this section, we describe our implementation of the homogeneous algebraic linear method, known as DLT, to solve a triangulation problem, that is, to determine the 3D coordinates of a point based on its projections onto two images whose camera matrices are known.

First, the correspondence between the 2D points in the two images, denoted as $\boldsymbol{x}$ and $\boldsymbol{x'}$, and the camera matrices P and P', must be known. In our case, putative correspondences are generated by randomly sampling 3D space and obtaining the corresponding projections onto two image planes. Usually, however, they are obtained by keypont matching pairs of images. Given a 2D to 2D correspondence, then, the goal is to find the 3D point $\boldsymbol{X}$ that satisfies Equation 1.

$$\begin{gathered} \boldsymbol{x} \equiv P\boldsymbol{X} \quad \text{and} \quad \boldsymbol{x'} \equiv P'\boldsymbol{X} \\ \text{where} \quad \boldsymbol{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \boldsymbol{x'} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}, P = \begin{pmatrix} \boldsymbol{p}_1^\top \\ \boldsymbol{p}_2^\top \\ \boldsymbol{p}_3^\top \end{pmatrix}, P' = \begin{pmatrix} \boldsymbol{p'}_1^\top \\ \boldsymbol{p'}_2^\top \\ \boldsymbol{p'}_3^\top \end{pmatrix} \end{gathered} \tag{1}$$

Note that the homogeneous scale factor $\boldsymbol{x} = \alpha P\boldsymbol{X}$ (implicit in the homogeneous equivalence) can be eliminated, as we are dealing with points in projective space.

Since $\boldsymbol{x} \equiv P\boldsymbol{X}$, their cross product is $\boldsymbol{x} \times P\boldsymbol{X} = 0$, which allows us to set up the following system of linear equations shown in Equation 2. However, the third equation is a linear combination of the first and second equations (x times the first equation plus y times the second equation) and thus we obtain just two equations per point. By following the same process for point $\boldsymbol{x'}$ in the second image, the problem formulation can be constructed as the concatenation of both 2D points as in Equation 3 such that $A\boldsymbol{X} = \boldsymbol{0}$.

$$\begin{aligned} y\boldsymbol{p}_3^\top \boldsymbol{X} - \boldsymbol{p}_2^\top \boldsymbol{X} &= 0 \\ \boldsymbol{p}_1^\top \boldsymbol{X} - x\boldsymbol{p}_3^\top \boldsymbol{X} &= 0 \\ x\boldsymbol{p}_2^\top \boldsymbol{X} - y\boldsymbol{p}_1^\top \boldsymbol{X} &= 0 \end{aligned} \tag{2}$$

$$\begin{bmatrix} y\boldsymbol{p}_3^\top - \boldsymbol{p}_2^\top \\ \boldsymbol{p}_1^\top - x\boldsymbol{p}_3^\top \\ y'\boldsymbol{p'}_3^\top - \boldsymbol{p'}_2^\top \\ \boldsymbol{p'}_1^\top - x'\boldsymbol{p'}_3^\top \end{bmatrix} \boldsymbol{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

However, solving for $A\boldsymbol{X} = \mathbf{0}$ is infeasible in practice due to measurement and imprecision errors in the point correspondences. Alternatively, to find the 3D point $\boldsymbol{X}$, the matrix A is subject to the least-squares minimization problem in Equation 4, with the constraint of $\|\boldsymbol{X}\|_2 = 1$ to avoid trivial solutions. This result can be achieved by computing the singular value decomposition of $A = USV^\top$ and taking the last column of V as $\boldsymbol{X}$, i.e. the singular vector associated to the minimum singular value of A. As an important remark, note that the DLT method is not invariant to similarity transformations, so a normalization step needs to be applied to the data before using it. This normalization prevents the effect of selecting an arbitrary origin and scale in the coordinate frame of the image. The matrix H shown in Equation 5, applies a scaling and translation so that both pixel coordinates of points are in the interval $[-1, 1]$. Instead of resolving the system of equations defined by $A\boldsymbol{X} = \mathbf{0}$, the normalized version $\hat{A}\boldsymbol{X} = \mathbf{0}$ is used, with $\boldsymbol{x} = H\boldsymbol{x}$, $\boldsymbol{x'} = H\boldsymbol{x'}$, $P = HP$ and $P' = HP'$.

$$\begin{aligned} \min_{\boldsymbol{X}} \quad & \|A\boldsymbol{X}\|_2 \\ \text{s.t.} \quad & \|\boldsymbol{X}\|_2 = 1 \end{aligned} \tag{4}$$

$$H = \begin{bmatrix} 2/n_x & 0 & -1 \\ 0 & 2/n_y & -1 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where n_x and n_y are the image width and height, respectively (note that by applying the same H to both images, we are assuming them to have the same size).

2.1 Results of the triangulation

In order to check the proper functioning of the implementation of the triangulation with the DLT method, we used the code provided as example and tested it using two cameras with known parameters and a set of random points. First, we computed the average triangulation error, and obtained a value of $\approx 1.31^{-15}$. Moreover, we plot both the ground truth and estimated points in the 3D space and show that the triangulation 3D points are nearly identical to their ground truth, as illustrated in Figure 1.

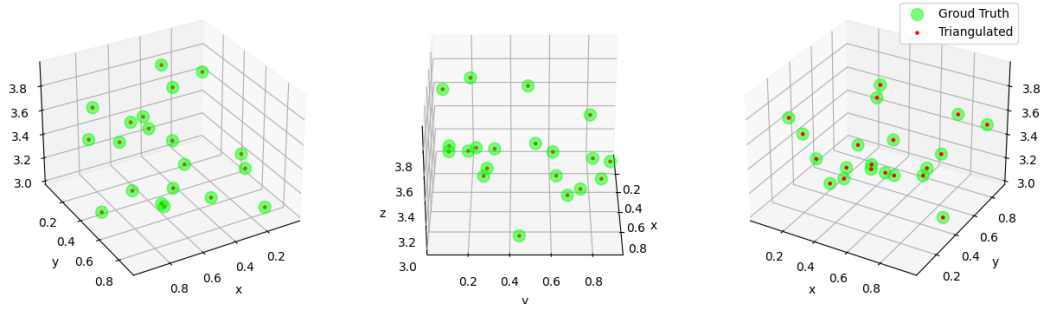

Figure 1: Comparison of the ground truth and estimated 3D points computed via triangulation with the DLT method.

Then, we test the effect that normalization has on this toy example. Thus, we do not include the normalization step and rerun the experiment. In this case, the average geometric error is $\approx 1.69^{-15}$, slightly above the previous value. The difference is small due to the controlled nature of the experiment, and the lack of measurement errors that real data suffers from. We do note, however, that the results have some randomness associated with them, as the sampled set of points varies each time. Nevertheless, the triangulation with normalization does seem to result in lower error in most cases.

	Geometric Error
With Normalization	$1.31 \cdot 10^{-15}$
No Normalization	$1.69 \cdot 10^{-15}$

In passing, we note that the derivation of the system of linear equations assumed corresponding points $x \leftrightarrow x'$ to have their third homogeneous coordinate equal to 1 (see Equation 1). Thus, we find it necessary to normalize the 2D points by the homogeneous coordinate before running the algorithm; otherwise, the results are extremely poor.

3 Reconstruction from two views

In this section, we tackle the 3D reconstruction from two views of a real-world scene in which the set of correspondences may be noisy and may contain outliers.

3.1 Estimation of the Fundamental and Essential Matrix

To do so, we first compute a set of putative correspondences between the two images. We use ORB keyponts and a brute-force matching procedure. The resulting matches can be seen in Figure 2a.

Then, we estimate the Fundamental matrix F relating the two views in 2D projective space (since the correspondences are noisy, we cannot directly estimate the Essential matrix relating 3D points.). To estimate F, we utilize the 8-point algorithm. To alleviate the noise in the data, we embed this algorithm into a RANSAC procedure that robustly estimates matrix F. Given this matrix F, we can then filter out outlier matches, ending up with a smaller set of good matches (Figure 2b).

(a) All keypoint matches found between images `Data/0001_s.png` and `Data/0002_s.png`.

(b) Inlier matches between images `Data/0001_s.png` and `Data/0002_s.png` after estimating matrix F with RANSAC.

Figure 2: Example of the use of RANSAC algorithm to remove outliers of keypoint matches between image views related by the fundamental matrix.

The next step is to compute the Essential matrix E. First we note that by the definition given by Faugueras et al.[1], the Fundamental matrix F is related to the Essential matrix E via the camera matrices K and K':

$$F = K'^{-T} E K^{-1} \Longrightarrow E = K'^{T} F K \tag{6}$$

Assuming both images are taken with the same camera, we have that the intrinsics are the same for both views, so we only have one calibration matrix ($K' = K$). In this case, $E = K^T F K$, and since we know both F and K, we can obtain E.

However, we still need to be careful about ensuring that E is a matrix of rank 2. In our code, we computed the SVD decomposition of the obtained E, and checked that the third singular component was 0 up to 9 decimal places.

3.2 Estimation of the Camera Matrices

The last step is to estimate the camera matrices from the Essential matrix. Given the SVD of E, $E = UDV^T$, four different camera poses for the second camera can be obtained as:

[1] Faugeras, Olivier D. et al. "Camera Self-Calibration: Theory and Experiments." European Conference on Computer Vision (1992).

$$
\begin{aligned}
P_1' &= \left[UWV^T | +u_3\right] \\
P_2' &= \left[UWV^T | -u_3\right] \\
P_3' &= \left[UW^TV^T | +u_3\right] \\
P_4' &= \left[UW^TV^T | -u_3\right]
\end{aligned} \tag{7}
$$

$$
\text{where} \quad W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } u_3 \text{ is the last column of U.}
$$

We have an ambiguity in terms of two camera orientations and two translations, ending up with four possible configurations. Only one, however, results in the scene points being in front of both cameras. Equivalently, only one of the configurations allows for the third coordinate of 2D points to be positive when obtained with projection P', the projection of the second camera.

Note also that the Essential matrix E is defined only up to a factor, which may be positive or negative. Since the rotations above are obtained via the U and V matrices from E's SVD, they also suffer from this ambiguity in sign. Thus, we are not guaranteed to obtain proper rotations satisfying $det(R) > 0$. Therefore, since we must deal with proper rotations to ensure that they conveniently form a group, we avoid the unwanted reflection in our code by flipping the rotation's sign if its determinant is negative.

Figure 3 illustrates the four possible candidates of positions and orientations for Camera 2 in relation of Camera 1. It is clear that the difference between each pair of solutions, (Cam 2_1: P_1', Cam 2_2: P_2') and (Cam 2_3: P_3', Cam 2_4: P_4'), is simply that the direction of the translation vector from the first to the second camera is reversed, as expected from Equation 7. The relationship of the first and third solutions is a little more complicated, but it can be verified that there is a rotation through 180º about the line joining the two camera centers. Two solutions related in this way are known as a "twisted pair" [2].

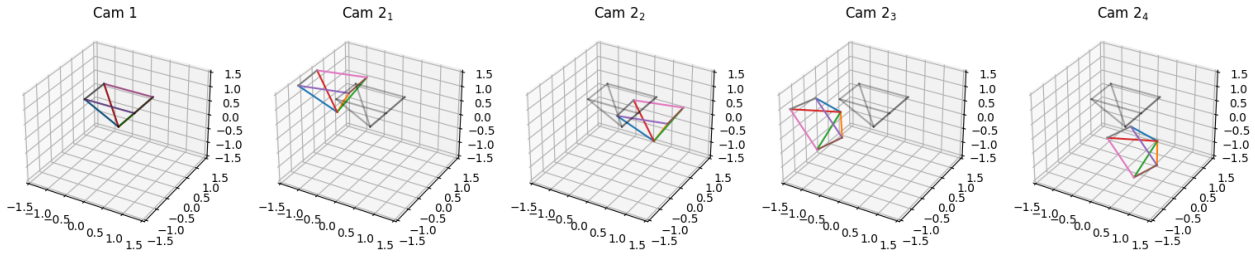

Figure 3: Visualization of the 4 possible candidates for Camera 2 in relation of Camera 1 for a stereo scene.

To select the optimal camera matrix P, we can follow two similar approaches. The first one we used was to select the camera P_2 such that the most points are in front of it. This can be computed by getting the sight line of each camera and counting the number of 3D points in front of them. The second approach we tested was to simply project the coordinates of the estimated 3D points into the camera coordinates for both views, and select the camera P_2 view if all the 2D points have a positive z coordinate for both cameras (again, meaning thy are in front of the camera). To better illustrate the above, Figure 4 shows each camera view candidate and the render of 3D points estimated by triangulation. The optimal camera matrix corresponds to Cam 2_3, in which practically all points are in front of both camera views.

[2] Hartley & Zisserman, 9.6.3

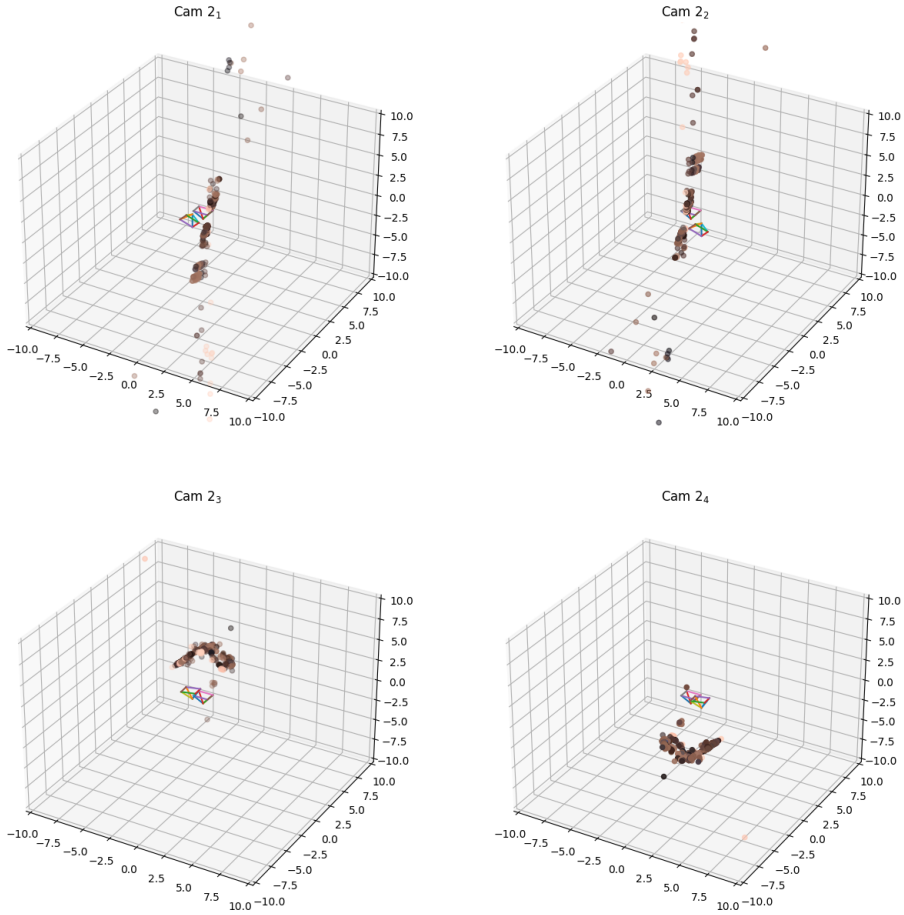

Figure 4: Visualization of the selection of the optimal P_2 camera matrix based on the configuration of 3D points.

3.3 Reprojection Error

The computation of the reprojection error is a crucial step in evaluating the accuracy of a 3D reconstruction quantitatively. The process involves projecting the triangulated points back into the image plane using the retrieved camera/projection matrix P. The error is calculated as the Euclidean difference between the projected points and the actual points in the image, as shown in Equation 8.

$$\sum_i d(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_i)^2 + d(\boldsymbol{x}'_i, \hat{\boldsymbol{x}}'_i)^2 \tag{8}$$

It is important to note that the triangulated points are obtained from noisy correspondences, and a zero reprojection error might not always be achieved. In our test shown in Figure 5, the results determine that although the error mostly concentrates on the lower values near zero pixels, it still has a non-zero value for some other bins. Also, the results shown in the table below indicate that the mean error is relatively high and that this method is sensitive to outliers. Moreover, due to the randomness of the RANSAC algorithm, the results and the error may change for each iteration.

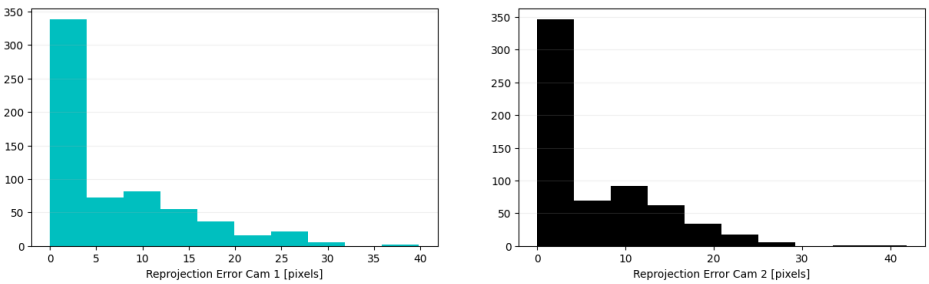

Figure 5: Histogram of the reprojection error for Cam 1 and Cam 2 views of the same scene.

	Mean Reprojection Error
Cam 1	6.6138
Cam 2	6.2612

4 Depth Estimation with Local Features

A potential way of inferring the 3D structure of a scene is trying to recover the per-pixel depth properties of the image. For doing so, in this section we explore an application for the stereo cameras set up. In this matter, we will try to use the local features of the image to explore the relative depth of each pixel by computing the distance of each point to its correspondence in the paired image. The intuition behind that is that closer elements present a bigger displacement in the stereo configuration, therefore, the depth of each pixel shall be proportional to the needed displacement for it to be aligned to itself in the paired image.

Note that we aim to compute disparity maps using local methods from pairs of rectified images. This rectification of images makes it easier to find correspondences between them. Given a point x in image I, the epipolar line l' in image I' with respect to x is parallel to the x-axis, meaning that the correspondence x' must be at the same height in the second image. This constraint simplifies the search for correspondences, as we only need to look for matches in the same row.

Thus, to find the correspondence for each pixel in the first image, we use a sliding window technique with a local metric. The search for the match in the second image is conducted in the same row by comparing the content of the reference window to the content of the other possible matches. Then, after evaluating the matching cost between the reference patch and all possible target patches, which are delimited by the minimum and maximum disparities, we pick the patch with the lowest cost through a Winner-Takes-All (WTA) approach.

The disparity between two corresponding points in a pair of rectified images is represented by the difference in their x coordinates. With rectified images, we have a parallel camera setup where the depth z of a 3D point X that projects onto points x and x' is related to the *baseline* and the focal length f as shown in Equation 9

$$\text{disparity} = x - x' = \frac{\text{baseline} \cdot f}{z} \tag{9}$$

4.1 Effect Of the Cost Function

Results may vary depending on the cost function indicating the distance of two regions assigned to the chance of both regions being the same. In this section we evaluate how this similarity function may affect the results of the algorithm.

Also, we provide a qualitative evaluation of each cost function computed with different sizes in the computed local areas.

4.1.1 SSD and SAD

As we observe in Figure 6 the qualitative difference of using the SAD and SSD cost functions is little to null. This may be due to the high correlation between both functions. Actually, if we compute the MSE and MAE costs with respect the groundtruth images we don't get any significant and consistent improvement when using SSD or SAD. For small windows sizes SSD slightly outperforms SAD For bigger windows, we get the contrary effect.

4.2 Effect Of the Weights

We also provide results with two additional adaptive weights, gaussian and bilateral. Surprisingly, as we observe in Figure 7 there is not significant improvement of using or not adaptive filters. Bilateral weighting, though, seem to provide 'smoother' results.

4.3 Test on Real-Scene Images

A real scene image is provided to test the method in a more accurate way. First, we set up the maximum disparity to 64 by empirical experimentation. We have to augment this maximum disparity due to two main reasons: first, as the image is bigger and more depth, a pixel displacement does not represent the same metric displacement than it used to do in the small image. Also, as there is a lot of repeating patterns and uniform areas, a small displacement can already fulfill the minimization of the cost function without an actual match. This second reason is why in Figure 8 we observe areas with some kind of repeating pattern instead of a uniform segmentation. As real-scene images have lot of repeating patterns, uniform regions... many times there is missmatching between this kinds of structures (for instance, a window in the facade can match with another window without being itself).

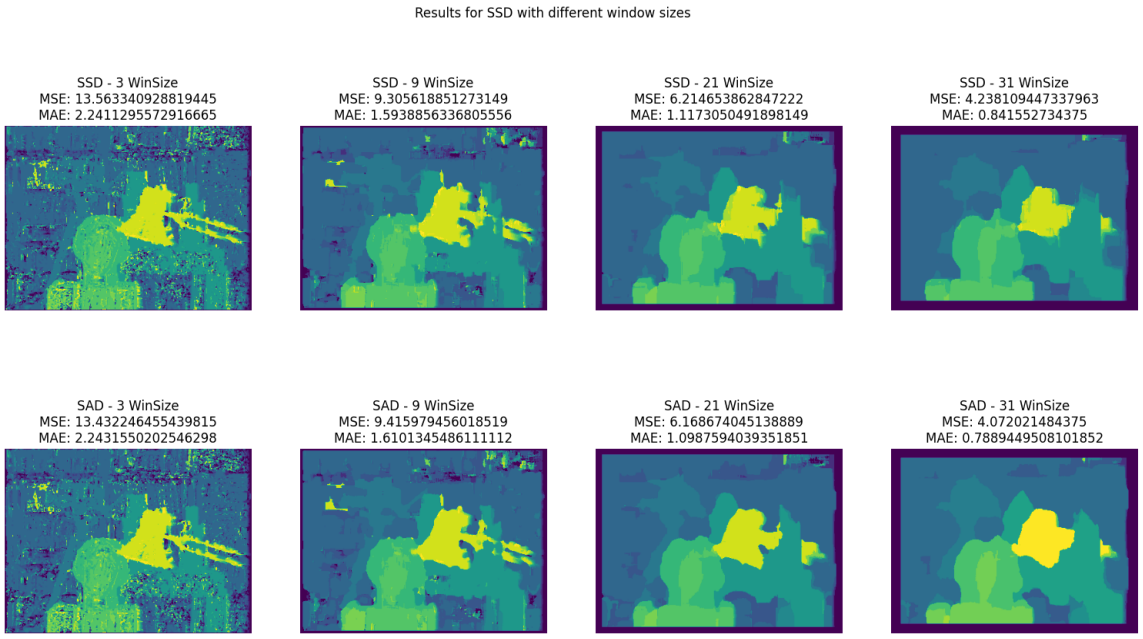

Figure 6: Results for the Sum of Squared Differences (top) and Sum of Absolute Differences (bottom) cost functions. Effect on the applied window size (horizontal).

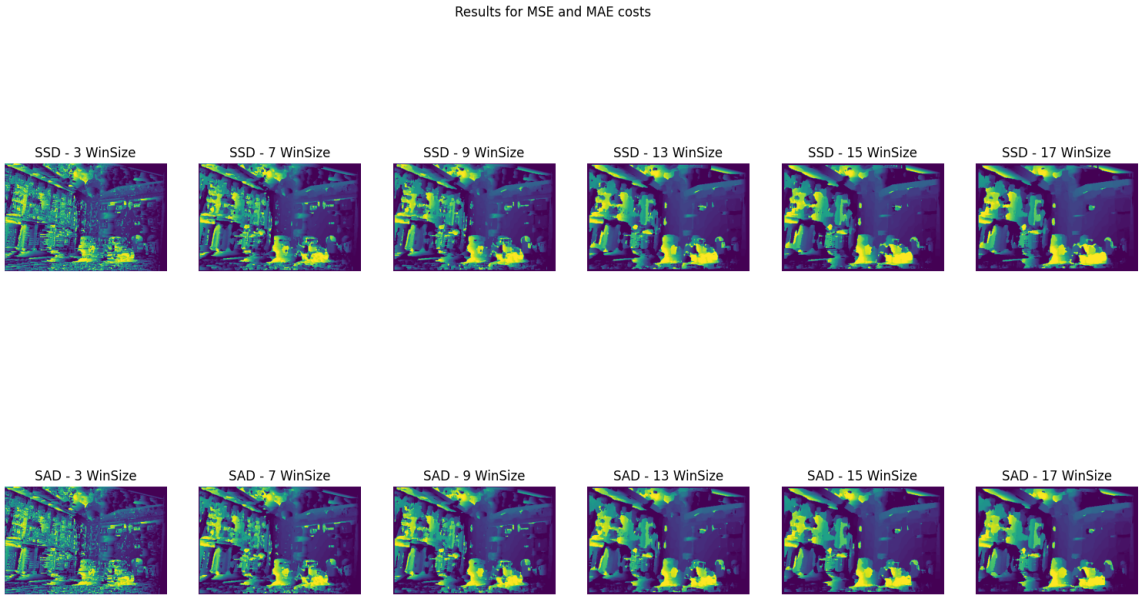

Figure 8: Results for the facade of the building. As we observe there's many missmatches in the areas with non-singular patterns.

5 Depth map fusion review

We review and compare two papers dealing with 3D reconstruction via fusion of depth maps: TSDF Fusion[3] and Routed-Fusion[4]. We begin by briefly summarizing each of them, after which we discuss their commonalities and differences.

5.1 TSDF Fusion

The method proposes fusing depth maps into an implicit volumetric representation, more precisely a cumulative weighted truncated signed distance function (TSDF) represented on a discrete voxel grid. Being an implicit representation, it offers high flexibility and expressiveness, at the cost of more expensive queries than explicit representations such as meshes.

[3]B. Curless and M. Levoy. A Volumetric Method for Building Complex Models from Range Images. In Proc. SIGGRAPH, 1996

[4]RoutedFusion: Learning Real-time Depth Map Fusion Silvan Weder, Johannes L. Schönberger, Marc Pollefeys, Martin R. Oswald May 2020

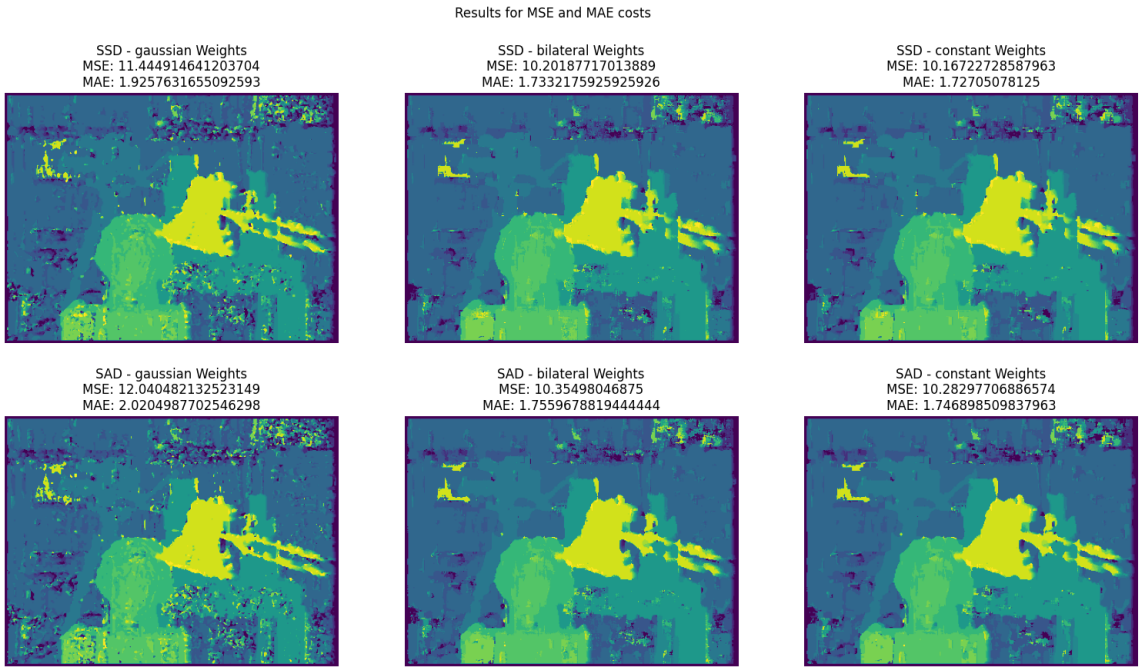

Figure 7: Results for the usage of adaptive weights. The windows size is fixed to 7.

The algorithm has two steps: first, a cumulative weighted SDF is obtained by combining the distance functions induced by each of the depth maps, weighted by some scanning technology-specific measure of the measurement's uncertainty (see Figure 9). This procedure however, only reconstructs the observed portions of the surface. Thus, the authors propose a second step to fill in the wholes in the reconstruction, by setting as surface points those lying at the frontier between unseen and empty regions (Figure 10).

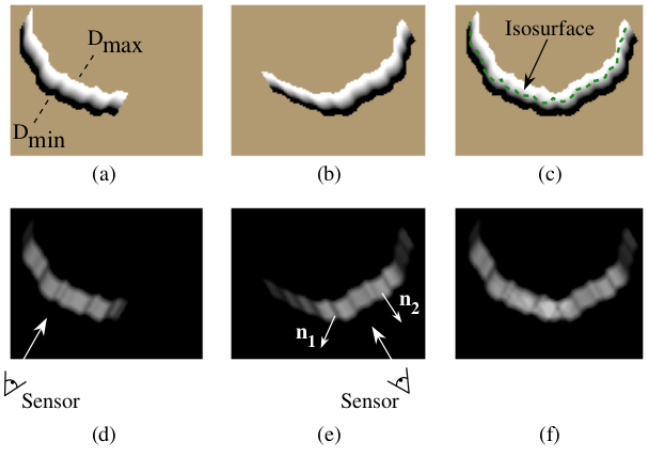

Figure 9: **Merging two signed distance and weight functions**. (a) and (b) are signed distance functions from two different views, with corresponding weight functions in (d) and (e). (c) is the SDF obtained from the per voxel weighted combination of (a) and (b), with weights from (f), which are obtained by summing the per voxel weights from (d) and (e). Note that camera intrinsics and viewpoint are assumed to be known, as depicted by the sensor in (d) and (e). Figure from the original paper.

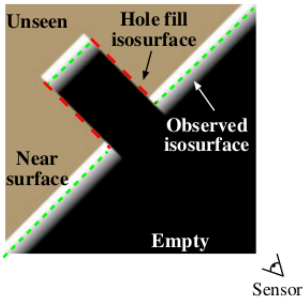

Figure 10: **Volumetric hole filling**. The regions in front of the surface are empty, while regions behind it remain unseen. The green segments are the surfaces are the 0-level isosurface, while the red segments are hole fillers, defined over the transition from empty to unseen space. Figure from the original paper.

5.2 RoutedFusion

This method largely builds upon TSDF Fusion. It adopts the same truncated signed distance function surface representation, and follows the incremental scheme for integrating depth maps introduced by Curless and Levoy. It also assumes know viewpoints and camera intrinsics. The crucial difference with respect to TSDF Fusion is that the update function for integrating new depth maps into the SDF is learned, rather than handcrafted. In addition to that, RoutedFusion (RF) presents a *depth routing* step that denoises and outlier-corrects input depth maps.

RF is a pipeline consisting of four essential processing steps, depicted in Figure 11. We will not go into each of these stages in detail. Rather, we will highlight the main differences and is similarities with the method by Curless and Levoy.

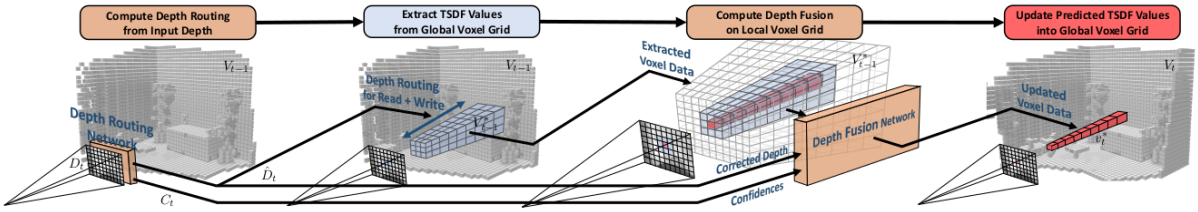

Figure 11: **RoutedFusion system overview.** The pipeline integrates depth maps into a global TSDF volume. A 2D Depth Routing Neural Network outputs the update location for every ray, given depth inputs. This module corrects noise and outliers, and estimates confidence values. Then, a depth and view-dependent local voxel grid is extracted. A Depth Fusion Network then predicts updates, given the local TSDF values, the depth and the confidences. The predicted TSDF values (red) are then written back into the global volume. Figure from the original paper.

5.3 Discussion

The most fundamental commonality between the two methods is the representation of the surface: both utilize a truncated SDF. The fact that the SDF is truncated allows for local updates and makes the method scalable. Though in different ways, both methods merge multiple noisy depth maps into a single robust estimate. Moreover, the TSDF is represented with a discrete voxel grid and the incremental update equations for this grid are the same in both works.

Overall, the most noticeable difference is that whereas TSDF Fusion implements a simple algorithm with handcrafted update rules, RF consists of a four-stage pipeline with learnable components. RF estimates for every ray the update locations of the voxel grid in a learnable fashion via a neural network, while TSDF Fusion just defines this update at all voxels in the grid. In this stage, RF corrects for gross outliers and missing values, which TSDF Fusion does not do. In addition, RF predicts the updates based on TSDF Values and depth information with a learnable neural network. Importantly, RF does not process each ray independently but allows for combining TSDF information from voxels around the ray. This differs from the approach taken in TSDF Fusion, where the update is given by a closed-form cumulative function, both for the weights and the distances, for each ray independently. Overall, RF learns a more robust weighting of input depths and carries out non-linear updates to more effectively handle noise, outliers, and thin objects.

In terms of efficiency, both methods are lightweight and can run in real time. In terms of quality of reconstruction, RoutedFusion performs much better on the ShapeNet dataset. In addition to that, the authors demonstrate how it is much more robust to noise than TSDF Fusion.

6 Conclusions

This week's project has focused on the topic of 3D reconstruction. We have implemented the triangulation method via the DLT, and tested it on a toy example, proving its efficiency for estimating Euclidean coordinates of 3D points.

The DLT method, however, assumes known camera projection matrices. Thus, to be able to apply them to real world scenarios, we must estimate these camera projections. Towards this goal, we use the 8-point algorithm to estimate the fundamental matrix and derive the essential matrix from it. Given the essential matrix, we can estimate the camera pose for a second camera, assuming the first one is at a canonical position. We realized that four possible choices of camera pose are possible, but only one of them leads to the scene points being in front of both cameras.

Thus, using triangulation, we were capable of generating a point cloud of estimations. Nonetheless, we noticed that due to the randomness in the RANSAC procedure for obtaining the fundamental matrix, the results were not too consistent across all executions.

Additionally, we devised an implementation of depth map computation with local methods and two different cost functions, namely, SSD and SAD. By generating the disparity maps, we are able to observe the different performances attained with each of the cost functions and window sizes. We realized that a smaller window leads to more detailed but noisier depth maps, while a bigger window smoothens the disparity at the cost of lost detail.

Lastly, we compared two methods dealing with depth-map fusion for 3D reconstruction: a learning-based approach featuring neural networks for non-local, non-linear updates of an SDF voxel grid; and a classic approach with handcrafted rules for iterative updates of an SDF voxel grid. Both techniques have their strengths. The learning-based one produces overall better results and is capable of handling missing values and outliers; the classic approach is much simpler and has already been tested in real-world applications. We note though, that both techniques are in principle light-weight and can run in real time. Moreover, due to their being based on the same underlying representation and incremental approach, the modern technique can serve as a quasi-drop-in replacement for the classic one.