3D recovery of urban scenes. Session 4

Guillem Capellera[1], Johnny Núñez[1], and Anna Oliveras[1]

1. Universitat Pompeu Fabra, Barcelona, Spain

0 Introduction

This session aims to implement a 3D reconstruction from two images with known internal parameters. We will use the Direct Linear Method (DLT) to triangulate the matching correspondences between two views of the same scene. We will also compute the camera matrices of the images given their intrinsic matrix and the fundamental matrix. To evaluate the triangulation method we will use the reprojection error. Then we will compute depth maps using local methods and we will study the effect of the window size and the use of bilateral weights.

1 Triangulation with DLT method

The goal of this section is to compute the 3D points corresponding to the matches of two images whose camera marices are known. The pseudocode of the triangulation using the DLT method is shown in Algorithm. 1. Note that in this case, we first need a normalization step but once we find the solution using the SVD decomposition we do not need the denormalization since the solution of the problem is the same as the original one.

To test the triangulation function we used some random points. First, we computed the canonical camera matrix for camera 1 and the camera matrix 2 using the rotation and the translation:

$P1 = [I|0] = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix}$, P2 = [R|t].

With these two camera matrices, we got the camera coordinates (rays) from each point and then we estimated the 3D points with the previous triangulation function (Algorithm. 1). We ensured that the triangulation function was correct since the average triangulation error was 1.6210−15, so we can consider it as if there is no error. In addition, in Fig. 1 we see that all the triangulated values are close to the ground truth.

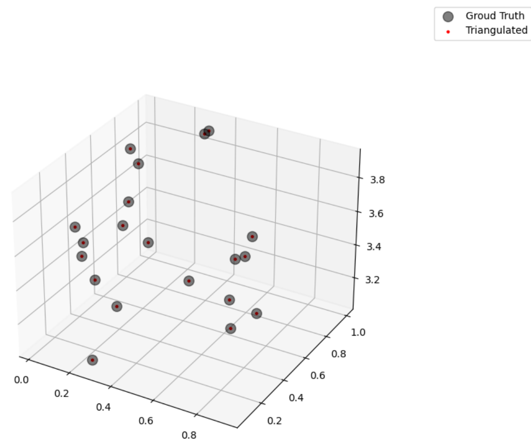

Fig. 1: Triangulated points vs ground truth

Algorithm 1 Triangulate

- **Inputs:** **x1**: matching points in projective coordinates from camera 1, shape (3, n_points)
  x2: matching points in projective coordinates from camera 2, shape (3, n_points)
  P1: camera matrix 1
  P1: camera matrix 2
  imsize: iterable with shape (2,) containing the image size
- **Outputs:** **X**: array containing the Homogeneous coordinates of the points in 3D space, shape (4, n_points)

1. Compute the homography H to normalize the points and the camera matrices so that both pixel coordinates are in the range [-1,1]. It is defined as
   $H = \begin{pmatrix} 2/nx & 0 & -1 \\ 0 & 2/ny & -1 \\ 0 & 0 & 1 \end{pmatrix}$, where nx and ny correspond to the image size.
2. Normalize the points and the camera matrices using H
3. For every point correspondences compute the system AX $=$ 0.
   Where $A = \begin{pmatrix} xp^{3T} - p^{1T} \\ yp^{3T} - p^{2T} \\ x'p'^{3T} - p'^{1T} \\ y'p'^{3T} - p'^{2T} \end{pmatrix}$ and X are the unknowns.
   Note that x and y are the coordinates of a point from the x1 set, x' and y' are the coordinates of a point from the x2 after normalization. Similarly, p corresponds to P1 and p' corresponds to P2 but they don't change between the systems.
4. To solve each of the previous AX $=$ 0 systems we do the Singular value decomposition of $A = UDV^T$ and the solution is directly the last column of V.
5. Construct the output X, of shape (4, n_points) with all the solutions of each system.

---

2 Reconstruction from two views

In this section, we estimate the 3D reconstruction from two views when the image correspondences contain outliers. To do so, we robustly estimate the fundamental matrix using the 8-point Algorithm and RANSAC. Once we have the fundamental matrix we compute the essential matrix and then we find the two camera matrices.

2.1 Estimate the image matches

The key points and descriptors are found using ORB and the matches are shown in Fig.2.

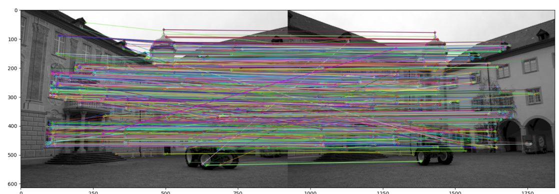

Fig. 2: ORB keypoints matches

2.2 Estimate the fundamental matrix

Similarly as in the previous session, to estimate the fundamental matrix (F) the 8-point algorithm in combination with RANSAC is used to find a robust estimation of F. The inlier matches are shown in Fig.2.

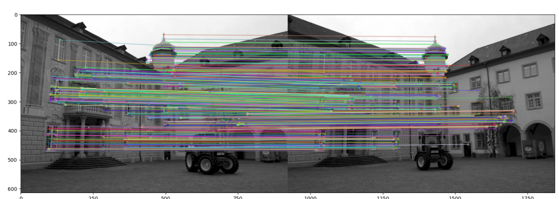

Fig. 3: Innlier matches obtained with RANSAC

2.3 Estimate the essential matrix

Once we have the fundamental matrix we can calculate the essential matrix using the camera calibration matrix K. So, the essential matrix is $E = K^T F K$ (since in this case K' = K) and it has rank 2, as expected.

2.4 Estimate the camera matrices from the essential matrix

We will assume that the first camera has a canonical matrix. Then we estimate the two possible rotation matrices doing the SVD of the essential matrix E. So, $E = UDV^T$ and the possible rotations are $R1 = UWV^T$ and $R2 = UW^TV^T$ where the auxiliary matrix $W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Note that we need to remove the unwanted reflections by multiplying R1 and R2 by -1 if their determinant is negative. This may happen because the rotation matrix represents the orientation of an object in 3D space, so it is possible to represent the same orientation with both a proper and an improper rotation matrix which is a reflection of the plane rather than a true rotation. There are also two possible translations that come from the last column of U with two different signs: $\pm u_3$. So, as we have two possible rotations and two possible translations, we have 4 options for the second camera matrix:
$P2_1 = [R1| + u3]$, $P2_2 = [R1| - u3]$, $P2_3 = [R2| + u3]$ and $P2_4 = [R2| - u3]$. Fig.4 shows the four mentioned options in relation to camera 1.

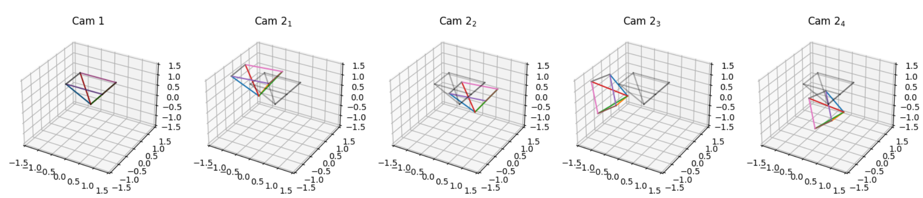

Fig. 4: Camera 2 in relation to camera 1

To select the optimal camera we will triangulate the correspondences. Of all the 4 solutions for camera 2, only one makes sense, since the reconstructed points should be in front of both cameras. Since there may be outliers, we will choose the camera that has more points in front of it (more points with component z positive). Fig. 5 shows the two cameras and the triangulated points. In this case the desired configuration is the first one, because it is the one that has more points in front of both cameras.

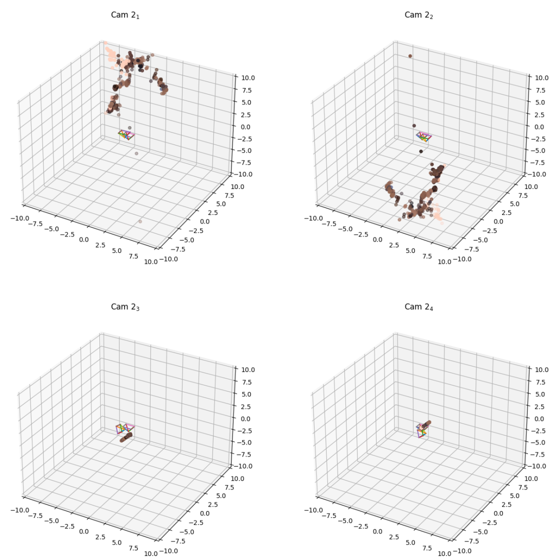

Fig. 5: Camera 1 and 2 and triangulated points for each of the possible solutions for camera 2

2.5 3D Visualization

Fig.6 shows the 3D coordinates for the matches for the selected camera. Note that almost all the points have a positive z.

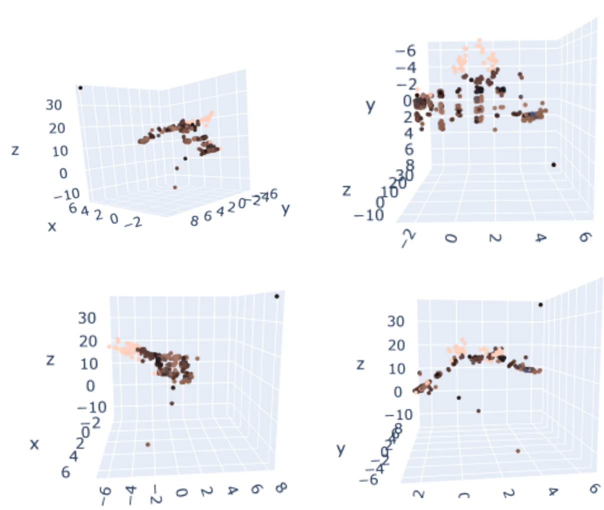

Fig. 6: Triangulated 3D points for the selected camera from different views

2.6 Reprojection error

We calculated the error of projecting the 3D space back to image coordinates to evaluate the accuracy of the estimated camera projection matrix. This error is the difference between the points and the estimated points in euclidean coordinates. The mean reprojection error in camera 1 is 0.63 [pixels^2] and for camera 2 is 0.58 [pixels^2]. Both mean errors are very low, other trials we have performed we obtained errors around between $8 \sim 50$ [pixels^2] for each camera. The histogram of each reprojection error is shown in Fig.7 and Fig.8, with different bin sizes. From figure Fig.8 we see that majority of the cases, the error is less than one pixel.

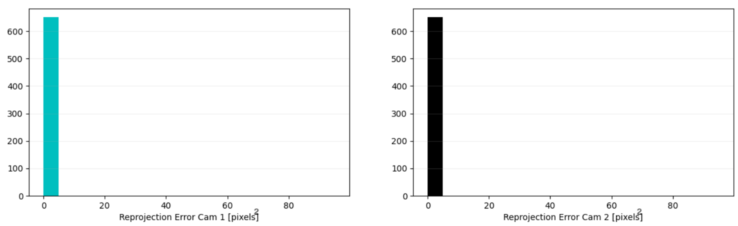

Fig. 7: Histogram of the reprojection error for each of the two cameras using bins of size 5

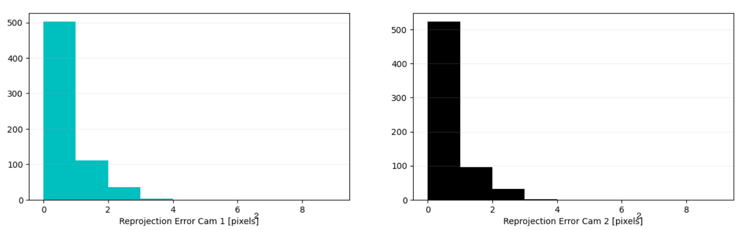

Fig. 8: Histogram of the reprojection error for each of the two cameras using bins of size 1

3 Depth map computation using local methods

Stereo matching involves finding matching pixels between two photos and translating their 2D positions into 3D depths in order to predict a 3D representation of a scene from multiple images.

In essence, we want to recover the depth at each pixel by using two camera positions to capture the same scene.

When two cameras are combined into a stereo system, the same view can be recorded, but their pixel points will have different coordinates. Disparity refers to the difference between these two sets of coordinates for the same scene location.

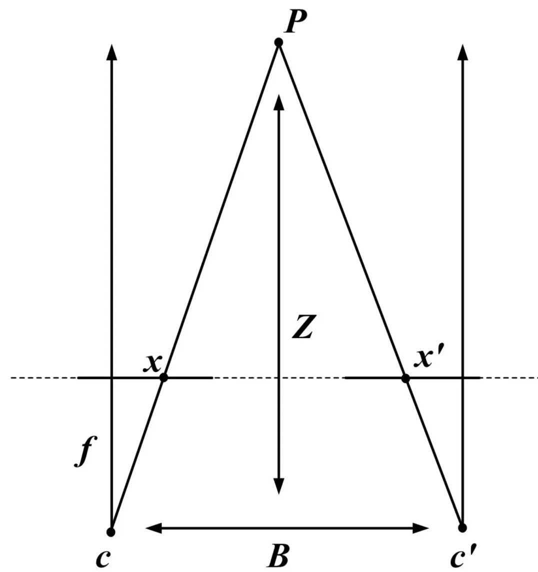

Fig. 9: Diagram of the stereo camera model from Wikipedia

As a result, the disparity is calculated as the distance between two frames between which the picture points have moved. By calculating the disparity for each matching pixel on the image, we will be able to produce a disparity map. Therefore, it enables us to recover the depth information and output the image's depth map, and computing the disparity map is crucial. Once the disparity has been calculated, the depth Z is defined as the distance between a point P in the real world and the camera. This diagram illustrates a stereo vision system with two parallel cameras, C and C'. Baseline is the distance B between the cameras, f is the focal length, and x and x' are the image planes of the cameras C and C'. As a result of triangulation, we can calculate the depth Z using the following formula, where (x - x') is the disparity. A pair of rectified images is subjected to this method, which involves sliding a window along the same line in the second image and comparing the content with that of the reference window. As a result, we select the pixel with the smallest matching cost (Winner-Take-All).

$$Z = \frac{f \cdot B}{(x - x')} \tag{1}$$

3.1 Vectorize

Vectorization allows array operations to be implemented without the use of for loops. By using highly optimized functions from different modules, we reduce the runtime and execution time of our code. Vectorized array operations will be faster than their pure Python counterparts in all types of numerical computations.

Python's for-loops are slower than those in C/C++. For the most part, Python is an interpreted language with a slow implementation rate. This delayed calculation is primarily due to the dynamic nature of Python and the absence of memory-intensive compiler-level optimizations. As NumPy is a C implementation of Python arrays, it provides vectorized operations on NumPy arrays.

The results are as expected with the tests performed.

Method	Absolut Result	Time (seconds)
Naive	0.362774	0.225146
Parallelized	0.362774	0.057522
Vectorized	0.362774	**0.00344**

(2)

3.2 Sum of Squared Differences cost

$$SSD(p,d) = \sum_{q \in N_p} w(p,q) \left(I_1(q) - I_2(q+d)\right)^2 \tag{3}$$

There are a number of methods to calculate similarity. The first is Sum of Squared Differences, which consists in calculating the difference between each pair of corresponding elements in two arrays and squaring them. Therefore, **Window with Minimum SSD = Most Similar/Matching Window**. In our case, we are not too familiar with vectorization, so we chose to parallelize. In this case, we went from 29 seconds with 1 core to 1.8 seconds with 16 cores.

Window Size	MSE
3×3	14.569
9×9	8.925
21×21	6.128
31×31	7.923

(4)

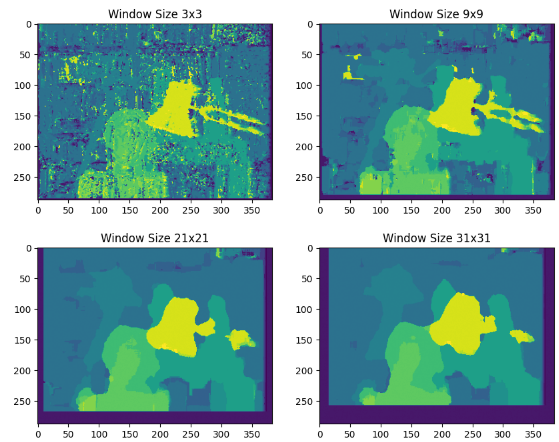

Fig. 10: Results of SSD-Stereo Matching

As we can see 13, it is evident that smaller windows have a greater degree of detail and are more sensitive to noise. Further, larger windows have the advantage of being robust to noise, but they are less precise and detailed, and they may also fail near boundaries. In the case of pixels with different levels of brightness, SSD cost function can result in very high costs.

3.3 Normalized cross-correlation cost

$$NCC(p, d) = \frac{\sum_{q \in N_p} w(p, q) \left(I_1(q) - \hat{I}_1\right) \left(I_2(q + d) - \hat{I}_2\right)}{\sigma_{I_1} \sigma_{I_2}} \tag{5}$$

The idea behind Normalized cross-correlation is as the intensity values of two patches may vary significantly, the correlation is normalized to take this into consideration. A standard deviation is calculated by subtracting the mean from each pixel value and dividing it by the mean. As a result, the result will be a value between -1 and 1, with 1 indicating a perfect match and -1 indicating a completely dissimilar match. Due to its fast computation speed and robustness to noise and changes in illumination.

Window Size	MSE
3×3	19.410
9×9	9.756
21×21	6.471
31×31	8.258

(6)

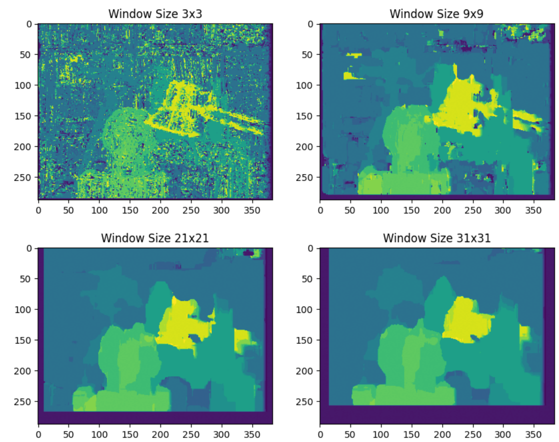

Fig. 11: Results of NCC-Stereo Matching

As we can see 6, the results are a little worse than SSD taking into account MSE.

3.4 Apply to facade images

Since the disparity between objects that are far away is small, changing the maximum disparity does not have a significant impact. The results obtained in this case are much worse than those obtained in the previous scene.

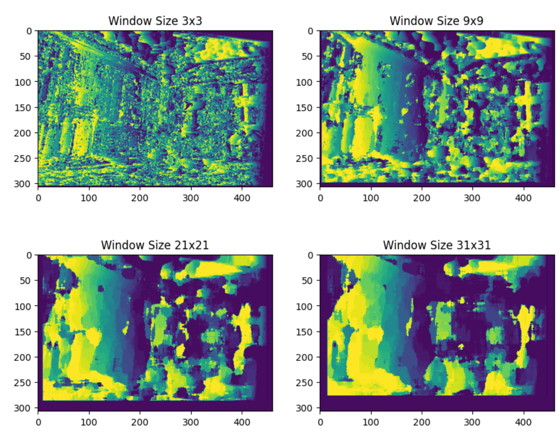

Fig. 12: Results of SSD-Stereo Matching

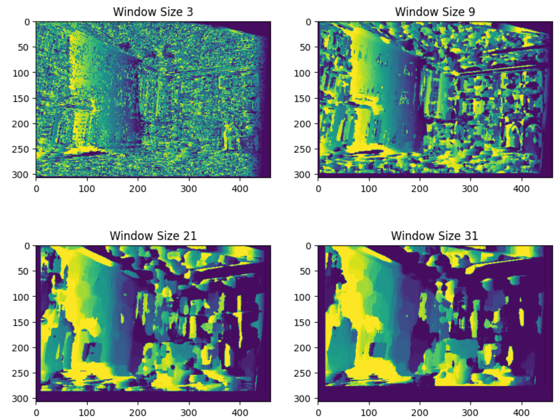

Fig. 13: Results of NCC-Stereo Matching

3.5 Adaptative support weights

The stereo_computation function, prior to the current modification, did not employ weights in its calculation of the cost between the reference and target windows. This was due to the assumption of a uniform distribution. To address this issue, the function has been modified to allow for the use of adaptive support weights (Yoon and Kweon, 2006).

Two functions have been developed to calculate two types of distributions: Gaussian and adaptive support weights. The Gaussian distribution assumes a normal distribution in the space dimension, while the bilateral adaptive distribution combines both the spatial distance and color difference. To compute the bilateral adaptive weights, the following expression, extracted from the class slides, has been used:

$$w(x,y) = \exp\left(-\frac{||I(x)-I(y)||_1}{\gamma_c} - \frac{||x-y||_2}{\gamma_p}\right) \tag{7}$$

where $w(x,y)$ is the weight between two pixels located at x and y, $I(x)$ and $I(y)$ are the intensities of the pixels at x and y, γ_c and γ_p are the parameters that control the influence of color difference and spatial distance, respectively. A higher γ_c value results in a higher weight associated with color. γ_p is proportional to the size of the window and is typically taken as the window's radius.

Figure 14 displays the various weight types discussed earlier. The first row shows the results for a window with a size of 60x60 using grayscale, while the second row shows the results for the same conditions using a color image (CIELab). The main difference is that in the color image case, the absolute difference between channels is first calculated, and then the mean is taken to reduce it to a single channel, resulting in the same representation as in the grayscale case. Utilizing adaptive weights incorporates local intensity information, allowing for the calculation of relationships (weights) between the center of the window and its surroundings. From Figure 14, it can be observed that the adaptive weights place almost all the importance on the lamp area. It is worth mentioning that uniform and Gaussian weights only depend on the window size and not its contents. This may be a suitable method if the color of the image is not essential in computing disparity. For the adaptive bilateral distribution, a γ_p equal to half the window size and a γ_c equal to 8 were used.

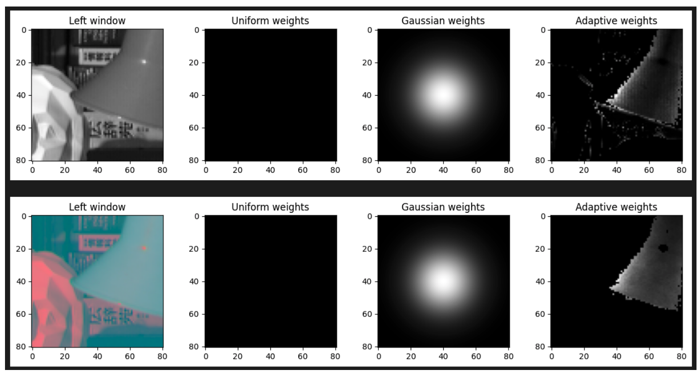

Fig. 14: The following figure illustrates an example of the different weights according to a certain window of the image reference.

The implementation of the Gaussian and adaptive support weights has been integrated into the stereo_computation function. The cost is calculated as a sum weighted by these weights. As shown in the Figure 15, both techniques, Gaussian and support adaptive weights, improve upon the uniform distribution measured using the SSD metric. We see that, for large window sizes, the adaptive allows us to visualize the objects in detail. This is enabled by taking into account the intensity of the pixels, which is the main factor of the adaptive technique. For the Gaussian distribution we used $\sigma = \frac{window\_size}{3}$ as it has to be related to the window size (similar to γ_p) . For the adaptive support weights parameters we have used $\gamma_c = 8$ and a $\gamma_p = \frac{window\_size}{3}$.

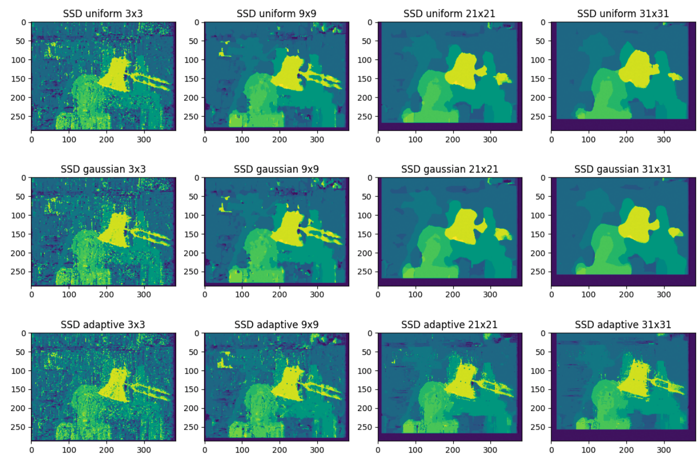

Fig. 15: Disparity map using different window sizes and different weighting methods: uniform, gaussian and adaptive.

Finally, in the Figure 16 we illustrate how γ_c value affects the disparity map, by fixing the window size in 31x31. In that case, results are quite similar but it can be seen that when this value increases the colour has more influence. In the other case, if γ_p adopts higher values than the ones we mentioned, the distance takes more importance and it can be understood as a kind of Gaussian non-adaptive weighting method.

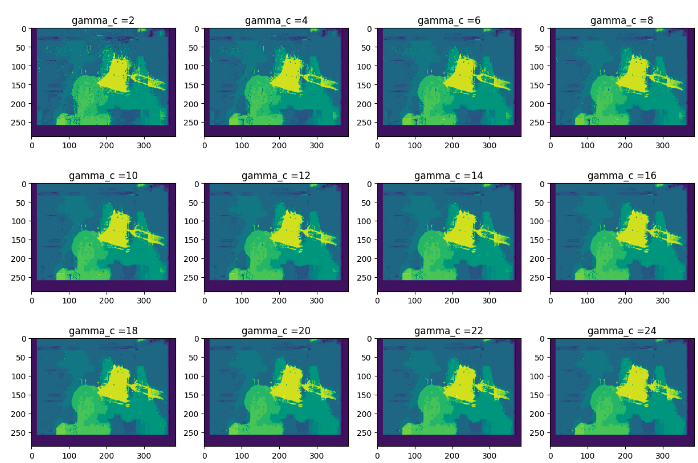

Fig. 16: Disparity map using different γ_c, which is a parameter related to the color influence when using adaptive support weights.

4 Optionals

4.1 Optional 2: New view synthesis using view morphing

In their paper "View Morphing", Seitz and C. Dyer, published in the Proceedings of ACM SIGGRAPH 1996, presents a method for generating intermediate views between two given views in a sequence of images. A technique called "view morphing" is proposed to create a smooth transition between two views, for example, for use in virtual walk-throughs, computer animations, and film special effects.

By using a set of control points, the paper discusses how to warp one view into another, transforming the geometry and appearance of each view. In order to generate intermediate views, control points are interpolated smoothly between the two views. Furthermore, the authors present a method for interpolating the intensity values of the pixels in the intermediate views, resulting in a seamless transition between the two original views.

In summary, the paper "View Morphing" presents a novel approach for generating intermediate views between two given views, using control points and interpolation techniques.

Throughout this section, we will interpolate between rectified stereo pairs of images from a scene and their corresponding ground truth disparities in order to generate new views of a scene. To form the equations below 8, we utilize the groundtruth disparity maps we have available. Considering that the images have been stereo rectified, the two pixels we interpolate lie at the same vertical level and we need to consider only the horizontal coordinate p in the morphed image for each correspondence pair.

$$\begin{aligned} \|d_l(x,y) - d_r\left(x - d_l(x,y), y\right)\| \leq \epsilon \\ \|d_r(x,y) - d_l\left(x + d_r(x,y), y\right)\| \leq \epsilon \end{aligned} \tag{8}$$

You can see the Gif in the folder. We generate 20 frames in total with 3 fps to smoothness, but it has cumulative errors during the generation along the generation of the images. The time to generate it is around 18 minutes. We try to vectorize or parallelize but we didn't have success.

4.2 Optional 3: Depth map fusion review

In this section, we will compare "the Volumetric Method for Building Complex Models from Range Images developed" (Curless and Levoy, 1996) with "RoutedFusion: Learning Real-Time Depth Map Fusion" (Weder et al., 2020). As opposed to the volumetric method, RoutedFusion uses a combination of neural networks and traditional computer vision techniques to represent 3D objects. In both cases, the objects are represented in 3D, but the techniques used to do so are different.

The paper "A Volumetric Method for Building Complex Models from Range Images" by Curless and Levoy describes a method for constructing complex 3D models from range images. The authors propose an efficient method for building, representing, and visualizing the model. An object is represented as a binary value in each voxel of this voxel grid using a voxel grid based on voxel grids. To construct a 3D model, range images are projected onto the grid.

Using neural network-based function approximators, the paper "Routed Fusion: Learning Real-Time Depth Map Fusion" presents a real-time depth map fusion approach. Convolutional Neural Networks (CNN) are used to predict depth maps for missing regions using depth cameras. The predicted depth maps are then combined with the original depth maps through the use of a routing algorithm. The routing mechanism routes information from depth cameras to predicted depth maps using a differentiable function.

Both papers aim to construct 3D models from depth maps, but their approaches and methodologies differ. Curless and Levoy's volumetric method is based on a voxel grid representation of the 3D model, whereas Routed Fusion approach is based on a neural network-based function approximator. Compared to the volumetric method, the Routed Fusion method is more modern and takes advantage of the latest advances in deep learning techniques. For example, the method uses a neural network to predict depth maps for missing regions, which improves accuracy. The result is a higher quality 3D model as compared to the volumetric method, which is limited by the resolution of the voxel grid. More advantages to the volumetric method, such as real-time performance, greater flexibility because it uses differentiable functions when routing information, and the ability to handle missing data, whereas the volumetric method requires complete depth data to construct a 3D model.

It is important to note that both methods rely on depth maps as input; however, the volumetric method projects the depth maps into a voxel grid, while the Routed Fusion approach uses a neural network to predict depth maps for missing regions. In the volumetric approach, the model is represented as a binary representation, whereas in the Routed Fusion approach, the model is represented as a continuous representation. Volumetric methods are limited in their ability to produce high-quality models due to the size of the voxel grid, whereas Routed Fusion approaches are able to produce models with a higher resolution.

Therefore, both the volumetric method proposed by Curless and Levoy and the Routed Fusion method are methods for constructing 3D models from depth maps. However, they differ in their approaches and methodologies. In contrast to the volumetric method, which is a traditional and computationally efficient approach, the routed fusion method is a modern and deep learning-based approach.

References

Curless, Brian and Marc Levoy (1996). “A Volumetric Method for Building Complex Models from Range Images”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New York, NY, USA: Association for Computing Machinery, 303–312. ISBN:

0897917464. DOI: 10.1145/237170.237269. URL: https://doi.org/10.1145/237170.237269.

Weder, Silvan et al. (2020). "RoutedFusion: Learning Real-Time Depth Map Fusion". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).*

Yoon, Kuk-Jin and In So Kweon (2006). "Adaptive support-weight approach for correspondence search". In: *IEEE transactions on pattern analysis and machine intelligence* 28.4, pp. 650–656.