Generating Point Clouds from Stereoscopic Images

Monday 6th March, 2023 - 08:54

Jason Billard
University of Luxembourg
Email: jason.billard.001@student.uni.lu

This report has been produced under the supervision of:
Jean Botev
University of Luxembourg
Email: jean.botev@uni.lu

Abstract

Computer vision is a widely researched field, with goal to make computers have vision. Vision is the ability to see the shapes and edges, recognise objects, and reconstruct the visual world. Stereo vision is a part of this field, focusing on interpreting the scene depicted in pairs of images, as humans do with their two eyes. In this BSP report written as part of the second semester of the BiCS program, a few computer vision algorithms are presented and a program generating a point cloud from pairs of stereo images is developed.

1. Introduction

This report presents algorithms used in computer vision to see the three dimensional shape of the scene with images, and presents a program developed as part of this BSP that converts stereo image pairs to point clouds.

The motivation for this BSP comes from the interest in computer vision. The pace of progress in computer vision has accelerated with the rise of machine learning, and the results continue to improve. Computer vision touches a large number of fields, all being used extensively in autonomous cars and augmented reality. Object and facial recognition [1], pose estimation, video tracking, 3d scene modelling are only a few domains of computer vision.

The fascinating results of 3d scene modelling that can be seen on google earth, in CGI in movies or as props in video games are build by scanning the scenes and objects with cameras and reconstructing the model with the images. The quality obtainable and the algorithms used motivate this BSP.

2. Project description

2.1. Domains

2.1.1. Scientific. The scientific deliverable covers depth perception in computer vision. Computer vision is a research field, in which methods are used to improve the extraction of relevant information from visual input such as images or videos.
This deliverable focuses on extracting the depth information from visual data.

2.1.2. Technical. The domains of the technical deliverable are computer vision and functional programming.
This project focuses on binocular vision, a field of computer vision where the visual input consists of pairs of images taken from a certain distance apart.

Functional programming is used to develop the program, which is a programming pattern to design the program around functions, rather than around classes.

2.2. Targeted Deliverables

2.2.1. Scientific deliverable. The scientific deliverable aims at answering the following question: What algorithms can be used to compute the depth of visual inputs (i.e. single image, set of images of a scene, videos,...) and to create a 3D model of the subject?".

The scientific deliverable is a comparison of different computer vision methods. The compared methods allow the extraction of depth information from one or multiple images and possibly additional position information.

For each method being compared, the underlying ideas behind the algorithms are described briefly. Their advantages and disadvantages over the other techniques are listed and the fields in which they are set out are described.

The first two methods are monocular vision algorithms. Such method relies on a single image to collect depth cues and construct depth information. The second method, binocular depth perception, obtains two images to extract depth information. The relative position of where the images where taken is also known, meaning it is similar to stereoscopic vision. Finally, the last method presented collects depth information from a set of images.

For each of the possible given inputs, there exists a large number of paper describing a way to solve for depth. In this scientific deliverable, the most common type of algorithms are presented, and other possible methods are hinted at.

2.2.2. Technical deliverable. As technical deliverable, a program will be developed which takes as input stereoscopic images and outputs a point-cloud corresponding to the scene photographed. The program is programmed using functional programming, and uses computer vision to generate the desired output. The the programming language used is Processing, a language is based on java, developed for a convenient programming of visual arts.
For this project, no computer vision library is used to simplify the process of generating the point clouds. The computer vision algorithms are implemented as part of the project, and their working will be explained thoroughly.
To generate the point cloud, the program creates a disparity map based on the two stereo images that are given as input. Using the disparity map, the depth is triangulated for each pixel on the image. To generate the disparity map, ideas and concepts of different papers are implemented.
The stereoscopic images are taken with a dual camera module connected over USB, and also with a mobile device by taking two pictures of the same subject from a different position. The program however can accept any arbitrary stereoscopic image pair.

An extension of the targeted deliverable would be to use multiple pairs of stereoscopic images of a same subject but from different angles. These pairs of images would be used to generate point clouds from the different viewing points. The point clouds are then merged to create a larger point cloud with the data of the different angles.

3. Pre-requisites

3.1. Scientific pre-requisites

Having a notion of the different time complexities will be required. In the comparison of the different algorithms to generate point clouds, the time complexity of the algorithm will be looked at.

3.2. Technical pre-requisites

As technical pre-requisites, a good understanding in three-dimensional geometry is necessary. This is required for the comprehension of the computation of the depth. Incidentally, some linear algebra is required for the explanation of the merging of point clouds using the ICP (Iterative Closest Point) algorithm.
On the other hand, being able to understand pseudo code easily is recommended to understand how the algorithm are implemented.

4. Scientific Deliverable

4.1. Requirements

This scientific deliverable will answer the questions "What algorithms can be used to compute the depth of visual inputs (i.e. single image, set of images of a scene, videos,...) and to create a 3D model of the subject?". It will be delivered as a text containing a list of different algorithms and techniques. For each technique, a description of their working is required.

For each algorithm, the input they use is precisely defined, and advantages and disadvantages are stated.

4.2. Design

To produce this deliverable, a number of research papers and a book have been picked and read partially. A section is given for some algorithm described in the papers and the book.

Four algorithms are presented in the production part of this deliverable. The first two algorithms use a single image as input, monocular vision. Then an algorithm which uses two images, binocular vision, will be described. And finally an approach to generate a point cloud when a set of images is used as input.

The first algorithm is called+ *Shape from Shading*, which reconstructs a mesh from the shading of the image. The second one is a learning based algorithm, which estimates the depth with a neural network. For the third, binocular vision has been chosen, which is also implemented in the technical deliverable. The last algorithm is implemented in the software Meshroom, and has a more developed pipeline.

These algorithms have been chosen for the reader to see the different approaches that can be taken to generate depth maps and 3 dimensional information from images. The algorithms have also been chosen to show the variety of approaches to solve similar problems in computer vision.

4.3. Production

Generating point clouds from visual input is called 3D imaging. Images are used and analysed to be able to generate a list of three dimensional points, corresponding to some point in the scene the image is from. The difficulty of this problem, is that images are a two dimensional projection of the three dimensional scene. One dimension of information is lost in the process of taking an image, and re extracting this information is the goal of this scientific domain.

The problem may however vary; sometimes the data acquired is only a single image, or it may be a set of multiple images from different cameras, from two cameras with known distance between them, a set of images with corresponding camera position, etc... For every input, a different algorithm is used to extract the most information out of the given data, to generate the most accurate point cloud of the scene.

4.3.1. Monocular vision. Monocular vision means that only a single image is used as input to the algorithm. With this single image, the algorithm has the extract depth information, to be able to generate a point cloud. As mentioned in the last paragraph, a single image is a two dimensional projection of a three dimensional scene, thus one dimension of information is lost, which has to be recovered out of the remaining data.

The reason as to why such algorithm can exist, is because the real world scenes have some general rules. For example, there are no points flying in the air, generally everything is attached and touching something else. Incidentally, shadows and colours can also be used to extract more information. These kind of principles could be used in algorithms to determine the depth of each pixel on the image.

4.3.1.1. **Shape from Shading**. Shape from Shading is an algorithm using the shading of the image of a surface to determine where the crevasses and hills are. A few constraints are set for this problem. The image given is a black and white image. This image contains a continuous surface lighten up by a light at the same position of the camera.

This algorithm uses the fact that the brightness of the surface is dot product of the normal of that surface and the direction of the light. In other words, the surface facing directly towards the light will be lit up fully, and the surface away or not directly at the light will be dimmer or entirely dark.

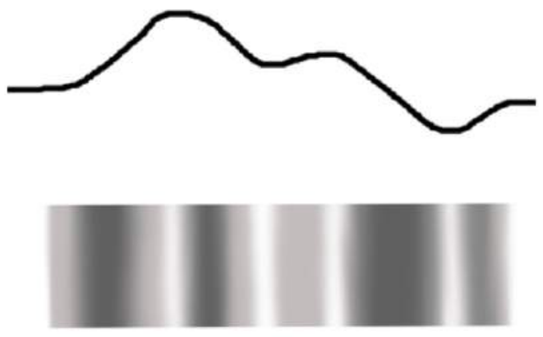


Fig. 1: Top: Shape of the surface, Bottom: Shade of the surface

This puts another constraint on the surface of the image; if the surface does not have a single colour, the variation on the darkness of the colour will describe a varying slope to the algorithm, which will then produce wrong depth estimations.

The greatest difficulty to overcome when recreating the shape from the shade, is that the shade indicates the absolute value of the slope. There is no information if the slope is going down or up, since it will get the same shade in both cases. This is called the *bas-relief ambiguity*, and is also present in human vision; some illusion take advantage of this advantage of this to make it difficult to distinguish hills from canyons.

There is no single technique trying to solve this SDS problem; and some of them are presented in the publication [3].

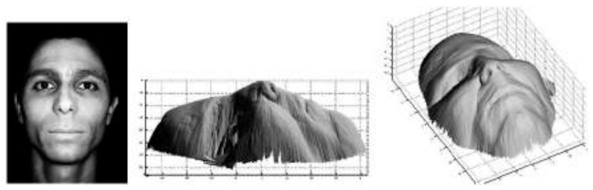


Fig. 2: Gray scale image of a face as visual input, and three dimensional shape generated with an algorithm described in the paper

4.3.1.2. **Zero-shot depth estimation**. More recent approaches to the problem of finding the depth of a single image are learning based approaches. Machine learning is a very broad topic, and in computer vision too, it is becoming a dominant tool for object recognition, feature detection, and many other fields, including estimating the depth.

In general, machine learning requires a training set - a database of input and expected results - which is used to train the model. In the case of estimating depth of images, the input is an image and the output a depth map, associating each point of the image to a depth.

A new paper [4], presented in 2020, develops on the idea of using a larger variety of data sets for learning. The larger and the bigger variety of the data used for training, the better the model will be able to estimate new depths. By being able to take advantage of multiple data sets, even if they have not been annotated the same way, this paper outperforms the other modern methods.

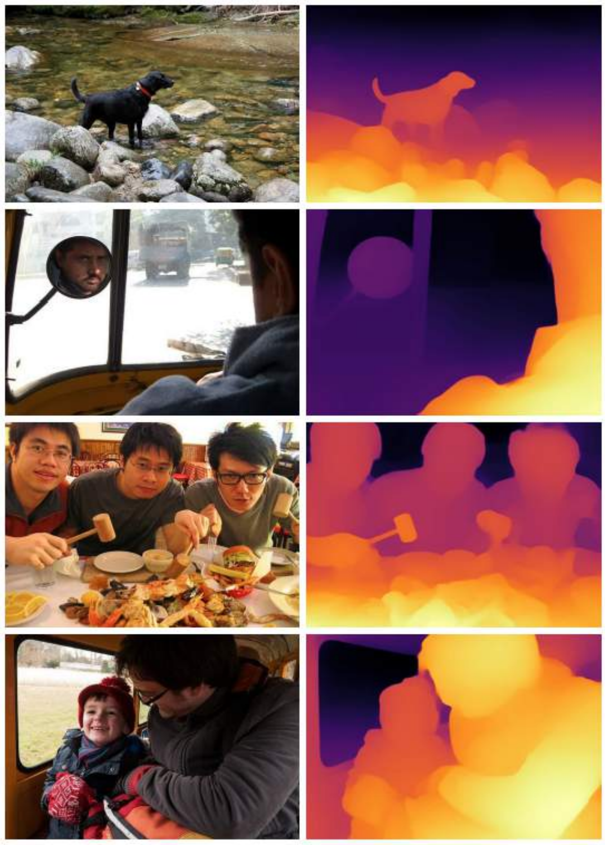


Fig. 3: Left: input image, Right: generated depth map with model in paper

The difficulty of using machine learning in this domain, is that there may not be any data in the data set that matches the input we present to the model. The model has to create an entirely new output, that it has never learned and which is not similar to anything created beforehand. This idea of not having an element in the data set that the model can ”remember” to generate a similar depth map is called *zero-shot*. Nevertheless, such approach can generate very precise outputs.

4.3.2. Binocular vision. Binocular vision uses two input images. The goal remains the same in all the different inputs; trying to extract as much information with the limited input. In this case, some other additional input might be given, which would allow for better extraction of data, for example the focal length of the cameras used to take these pictures and the distance between them.

An binocular vision algorithm is developed in the technical deliverable of this BSP. However, the requirements of the inputs are stricter than general binocular vision. In most cases, cameras are not calibrated, and it is difficult to make them point in the exact same direction. Because of these distortions and imperfections, camera calibrations become an important topic when more than 1 image is used as input.

In the calibration process, the distortions like fisheye effect are removed, and the epipolar geometry is found (lines on different images corresponding to same points in scene [5]). After the calibration, the images are transformed so that the epipolar lines are horizontal. A horizontal line in one image then corresponds to the same line in the other image.

These calibrated images are used as input in the technical deliverable. To find the depth, the pixels on both images corresponding to the same point in the scene are matched. As is explained more deeply in the design section of the technical deliverable, the distance between this two pixel correlates to how close the point is to the cameras.

This technique used in the technical deliverable can extract information about the depth of the pixels, using information given by both images.

Binocular vision is used to rapidly and easily model 3d scenes; with a hand held stereo camera, one can screen the scene efficiently.

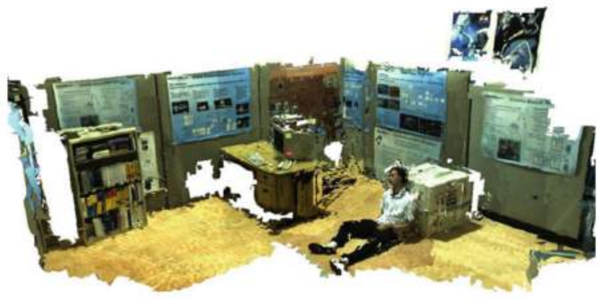


Fig. 4: Scene recreated using hand held camera with method proposed in [6]

4.3.3. General 3D Imaging. In this section, a set of images of some scene is given as the input, and the goal is to recreate a 3 dimension representation of this scene. The algorithm presented in this section uses feature detection to find key recognisable points in the scene, and finds the images of these key points in the different images. With the information of where these features are located in the image, the algorithm can find the position of the cameras, and then easily densify the point cloud using something similar as binocular vision.

The method being presented [7] is used in Meshroom to generate three dimensional meshes from sets of images.

The first step in the pipeline of this software is to extract key features that can be recognised easily. The most common feature system is SIFT [8] (Scale Invariant Feature Transform), which is used along other feature systems in this program. Features are areas in the images, that can easily be recognised in other images.

Images that have some matching features are then put together, and the features are matched between these images.

These feature matches are then used to build an undetailed model of scene.

At this point, the software knows the position of the cameras relative to each other and to the feature matches, with which the scene was build upon. Now, a depth estimation is performed on all images. However, only the parts which overlap with other images can be estimated. To perform this estimation, a grid of cubes (voxels) are is used. The voxels are filled depending on whether or not their existence makes the depth map more accurate. When a voxel is filled, the depth map of different camera views changes, and the accuracy of the voxel representation of the scene can then by checked according to these point of views. This software starts with a grid with low details, and iteratively increases the details.

Since this software creates a mesh and not a voxel representation, a point cloud is extracted from the depth maps and then a mesh is build from it. The images are then projected on the mesh to generate the texture.

This photogrammetry software can generate highly accurate meshes from a set of images.

4.4. Assessment

The variety of the algorithms is present because of all the possible requirements and restrictions that can be set. Some algorithms produce better quality output, but require more processing time and power. A different solution is proposed depending on the requirements and restrictions. The processor which will need this algorithm may have more or less processing power. It may be that the camera with which the input image is produced is only in black and white, or may have a very low resolution. Meanwhile, some system may be able to take a large set of high quality images, and require the output to be of very high quality and be produced fast. The algorithm also varies depending on the type of images, if it is close up images or satellite images, whether or not the image is always consistent.

The list of parameters is very large, which is why this field is one of the most researched and will continue to be researched, proposing new interesting and more developed algorithms every year.

5. Point Cloud Generation from Stereo Images

5.1. Requirements

The requirements of this project is to develop a program that generates point clouds from stereo images.
The precise input of the program is a single image, composed of two images. The images are a pair of stereo images. The images are then put next to each other horizontally to form the input image. The height and width of the left and right stereo images are equal.

Stereo image pairs are two images that satisfy certain conditions. The image plane of both images is the same, which means that the images are pointing in the same direction. The change of position of the camera can only be horizontal, meaning that both images are taken as the same height and a certain distance apart. The images should be taken with the same or an equivalent camera. This distance is given as input to the program, as well as the focal length of the camera.

The device that is used to take the pictures should not have any kind of noticeable distortion, such as fish eye effect. Incidentally, the images should be taken at the same instance for dynamic scenes. For static scenes, images may be taken at different time, but shall have the same lighting conditions.

The program takes the input image containing both stereo images, as well as the distance between the images coordinates and the focal length of the camera. After processing the image, the program returns a point cloud containing a set of coloured points representing the scene that is displayed in the stereo images.

The program is written in Processing, which is a programming language based on Java but simplifying the development of visual arts.

The program should take a reasonable amount of time to generate the point cloud, that is it should not need hours to process a 1 megabyte image.

5.2. Design

In this section, the steps taken to develop the technical deliverable are discussed.

To develop the technical deliverable, the programming language Processing has been chosen for several reasons. Processing has been developed with visual arts in mind, and facilitates displaying images and other visuals. As this programming language is based on Java, it offers the same flexibilities as Java does, in addition to what Processing add, such as functions without classes.

The process of generating the point cloud from the input images is divided in several steps. Each step is dived in a subsection for a clearer overview. The first step is cutting the starting image in half. Then a disparity map is generated, however it is noisy. It is then post processed to remove as much noise as possible. The program then uses the focal length and the distance between the cameras to compute the depth of each pixel on the image. Using the depth map and the focal length, the program projects the image back in the third dimension.

5.2.1. Cropping the Image. The input image is an image containing two images of same with attached together horizontally. To distinguish between the two images, the program cuts the image in the middle and forms a left and right image.

5.2.2. Computing the Disparity Map.

5.2.2.1. **Definition of Disparity Map**. The disparity map of two stereo images is a map containing for each pixel how much they shift horizontally in comparison to the other image. The disparity map can thus be represented as a matrix containing values with unit ”pixels shifted to the right”.

It is important to note that some pixels might be visible on one image but not the other, thus it is possible that the value in the matrix is not defined. Each pixel on the left image have a corresponding pixel on the right image, which can be found by looking at the disparity map and looking at the right image n pixels to the right, where n is the value of the disparity map at the location of that pixel.

5.2.2.2. **Minimising Cost of Disparity Map**. To measure how good the disparity map is, a cost function is introduced. This cost function can measure how similar pixels are to each other. It receives the coordinates of two pixels, one of the left image and one of the right image, and returns a low value if the pixels are similar. Conversely, it returns a large number if the pixels are not similar.

Now that a cost function exists, the goal is to reduce the cost of the disparity map to a minimum. The cost of the disparity map is the sum of the cost of all pixels with their corresponding pixels on the other image. How this is done is explained in the following sections.

The most primitive cost function would measure the distance of the colours of the two pixels. However, such a cost function is too basic, and does not capture enough information to be a reliable way to compute disparity maps. Thus, a more developed version of this method is used. A window is chosen, in the context of this technical deliverable, only square windows will be considered. A 5×5 window would have the targeted pixel at the centre, but also include pixels 2 above and under as well as left and right of the current pixels. The cost function is the sum of the cost of all the pixels in the windows with the corresponding window. Note that the entire window is moved by the disparity of the targeted pixels, and not individually.

5.2.2.3. **Finding the Disparity of Pixels**. Finding the disparity of a pixel is synonymous to finding the corresponding pixel on the other image. After finding two matching pixels, the disparity is their horizontal difference.

The cost functions allows the measurement of similarity. To find the matching pixel on the other image, it suffices to take the most similar pixel.

Using this idea, an algorithm comparing every pixel on the left image to all the pixels on the right image can be created. This program then takes the pixel with the lowest cost - the highest similarity - and considers it as the corresponding pixel. If the cost function was perfect, this would theoretically produce a perfect result.
The speed of this algorithm however is a drawback. For a 1 MP image, this image would compare $10^{6^{10^6}}$ pixels. This is considering that the window size is 1. For an image with n pixels, this algorithm has a complexity of $O(n^n)$.

A few constraints can be set to reduce this time complexity. It is known that the stereo images are taken with camera without distortions, and the difference of the coordinates of the camera when taken the images is strictly horizontal. Thus, a pixel at a specific row of the image will also be at the same row on the other image. With this, only the pixels in the row of the targeted pixels have to be checked. Incidentally, if the camera is moved to the right, all the pixels on the image will certainly move to the left. How much they move depends on their depth, a fact used to compute the depth later on. This information reduces again the number of pixels that have to be checked.
A maximum disparity can also be set. Setting a maximum disparity assumes that all the pixels have more than a certain depth. This is reasonable to assume since pictures are usually taken at a certain distance from the subject, and the camera is not touching the scene.
Taking these constraints in account, the number of pixels to match with is reduced to only the maximum disparity, setting the time complexity to $O(n)$ which is a significant improvement.

In this technical deliverable, another improvement is developed: the stereo images are first scaled down and a disparity map is generated using the previously mentioned technique. Then, this disparity map is scaled up again, and used as a base to compute the disparity map of the full size images. This kind of recursive disparity map generation can be repeated multiple times, but the best results have been created by performing this operation only once. How this base disparity map is used will be explained in the technical part of the deliverable. This method manages to divide the processing time by a significant factor.

5.2.3. Post Processing: Denoising the Disparity Map. In contrary to what has been assumed in the previous section, the cost function that is used to measure the similarities of pixel is not perfect. This causes noise in the disparity. In this technical deliverable, the noise is divided in two different categories, high frequency noise and low frequency noise. Both types of noise are treated differently.

- High frequency noise are single pixels that have wrong disparities, surrounded by pixels with accurate disparities. It is caused when the cost function inaccurately describes a pixel to be similar to another, even if it is not.
  To distinguish between correct disparities and high frequency noise, the neighbour disparities are compared. If there are enough neighbour pixels with a similar disparity, this pixel is not high frequency noise. If it is, that pixel can then be fixed by setting its disparity to the disparity of a neighbour that is not itself noise.
- Low frequency noise are larger groups of pixels with similarly wrong disparities. They are often found in groups, on large flat areas with similar colours, or in repeating patterns. It is caused when areas of pixels are similar (the cost function does not perform well on large areas with a single colour), or when the base disparity map has errors.
  Recall the method used to distinguish high frequency noise. This same method cannot be applied in the context of low frequency noise, since they come in patches of similar disparity pixels. A more developed algorithm is required to find these pixels.

  A method has been developed as part of this technical deliverable, which can detect low frequency noise. It works with the same principle as the method used to detect high frequency noise. This algorithm iterates in a circle around the targeted pixel, in search of a pixel with similar disparity (that is, a pixel with a disparity difference smaller than some threshold). The radius of the circle determines how large a patch of pixel with different disparity is considered noise. With a very small radius, only higher frequency noise is detected. However, when the radius is set too large, pixels with accurate disparities might be wrongly detected.
  After detecting these pixels, the same method to correct high frequency noise is used to correct them low frequency noise.

In the technical deliverable, other methods that have been tried to reduce the noise and improve the quality of the disparity map will be explained. However, they are not described in the design section, as the results of these methods were undesirable, and are not used in the final program.

5.2.4. Post Processing: Smoothing out the Disparity. When generating a disparity using the previously mentioned algorithm, the disparity remains an integer value. This is an issue, since when this will be converted in a point cloud, the points will be separated with layers. This is due to the non-continuity of integer number.

To prevent this, the disparity map is smoothed before the point cloud is generated. A naive approach to smooth the disparity map is to perform a Gaussian blur. However, this will also smooth out edges, and a lot of information will be lost.
Another method would be to perform a blur, but only taking into account similar disparities. This will keep the separation of regions with large disparity differences sharp, but still smooth out the regions with similar disparity.

5.2.5. Computing the Depth with the Disparity. After passing through the previous steps, the program has a map, which contains the disparity of each pixel. In this step, the program uses the effect known as parallax to find the depth of each pixel. By definition [2], ‘Parallax is the effect by which the position or direction of an object appears to change when the object is seen from different positions.‘

In the context of stereoscopic images, the parallax is used since objects closer by will move more than objects further away. Hence, if the disparity is high, it means that the pixel moved a lot from one image to the other, resulting in a closer pixel.

In the following part, the formula to compute the depth according to the focal length of the camera, the distance between the cameras and the disparity will be derived.

Let f be the focal length of the camera and d the distance between the cameras. The disparity is noted with k and the width of the image be w. A camera of width 1 is assumed. To simplify the calculations, the position of the pixel relative to the centre of the camera plane (in the representation used in this section, it is only a line as the height axis is not taken into account) is used instead of the disparity.

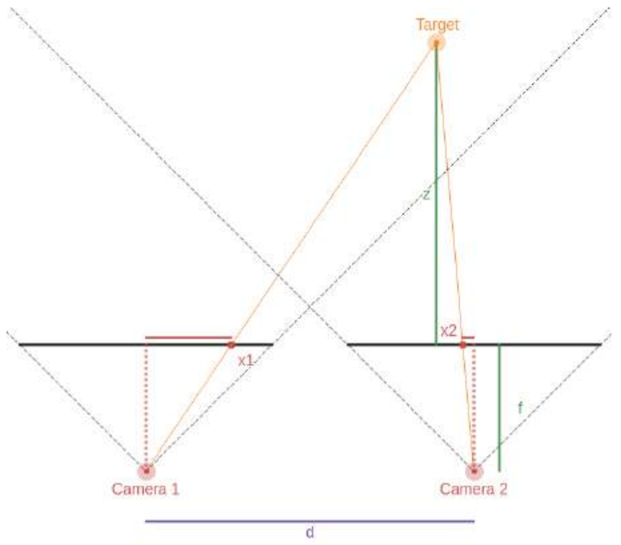


Fig. 5: *Two camera with distance 2 between each other. x1 and x2 lay on the image plan and are where the target is displayed on the image. f is the focal length and z the depth of the target.*

Note that the disparity k is equal to $(x_2 - x_1)w$. *Target* is the point that is displayed by the pixels currently being analysed.

Two triangles can be formed from the cameras origins with the target point. They are formed in the following manner:

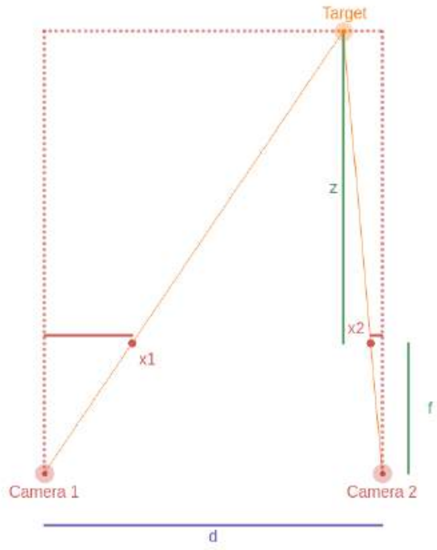


Fig. 6: *Two right triangles formed with the two cameras, the target, and the points at the horizontal position of the cameras and depth of the target. x1 is the horizontal position of the target on the image plane of camera 1, and x2 is the same but on camera 2. d is the distance between the camera, z is the depth of the target, and f the focal length.*

The angle formed by these right triangle can be computed using the invert tangent function:

$$\begin{aligned} a_1 &= tan^{-1}\left(\frac{x_1}{f}\right) \\ a_2 &= tan^{-1}\left(\frac{x_2}{f}\right) \end{aligned} \tag{1}$$

In figure 5, a_2 is a negative angle, since x_2 is left of the centre of camera 2.
Both triangle touch at the target, thus summing the length of the opposite sides of the triangles equals to the distance between the cameras d. It is important to remember that the direction in which the triangle points is considered. In this context, when talking about length, it is not always a positive value.

The following equality is derived:

$$\begin{aligned} & (z+f)tan\left(a_1\right) = d + (z+f)tan\left(a_2\right) \\ \iff & (z+f)tan\left(a_1\right) - (z+f)tan\left(a_2\right) = d \\ \iff & (z+f)\left(tan\left(a_1\right) - tan\left(a_2\right)\right) = d \\ \iff & z+f = \frac{d}{tan\left(a_1\right) - tan\left(a_2\right)} \\ \iff & z+f = \frac{d}{\frac{x_1}{f} - \frac{x_2}{f}} \\ \iff & z = \frac{df}{x_1 - x_2} - f \end{aligned} \tag{2}$$

Replacing the denominator of the fraction with the disparity, we obtain:

$$\begin{aligned} z &= \frac{df}{-\frac{k}{w}} - f \\ &= -\frac{dfw}{k} - f \end{aligned} \tag{3}$$

In the program, a map is formed containing the depth for each pixel.

5.2.6. Creating the Point Cloud from the Depth Map. With the depth map, the program now has to generate a point cloud. For each pixel, the program, will use the depth map to find out at what depth the pixel is located, and project it back into 3D space and also assigning a colour to that point.

As the camera frustum is not a rectangle, rather a pyramid, the x and y coordinates of the point in space are not equal to the x and y coordinates of the pixel on the image. To find the coordinates, the same formula is used as to compute the depth. However, this time the unknown variable is the length of the opposite side, and the depth and angle are known.

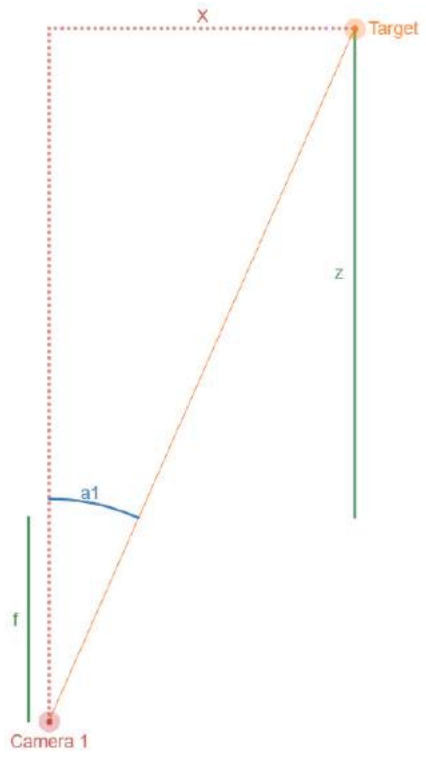


Fig. 7: Right triangle. a1 is angle to the target pixel. X horizontal distance of the target point with the camera. f is the focal length and Z is the depth between the target point and image place.

The following formula can be derived:

$$X = tan(a_1) \times (z + f) \tag{4}$$

For the Y coordinate of the point, the same formula can be applied, except using the vertical position of the pixel. The only difference is that the aspect ratio of the image is also a factor, as the width of the image is assumed to be 1, but not the height.

The depth is known, as it was computed in the previous section. Finally, the colour associated to that particular pixel is assigned to the point. This is repeated for every point.

This results in an list containing all the points in the point cloud. To visualize the point clouds, it can be displayed using processing or exported in a file to be viewed in a more specialized software.

5.3. Production

To present how the technical deliverable has been produced, the production section is divided in the following way. First of all, the main pipeline of the program will be described. Then the implementation of the important step in this pipeline will be described in their own subsection.

The images used in the production section are from the Middlebury datasets.

5.3.1. Program Pipeline. The program follows a specific pipeline to produce the point cloud. In the following code snippet, the main steps of the point cloud are described.

```
PImage image = loadImage(stereoImagePath);

PImage left = cropImage(image, 0, 0, image.width / 2, image.height);
PImage right= cropImage(image, image.width / 2, 0, image.width / 2, image.height);

float[][] disparityMap = iterativeDisparityMap(left, right, 3);
float[][] denoisedMap = denoiseMap(left, disparityMap);
float[][] smoothedMap = smoothMap(left, denoisedMap);

ArrayList<Point> pointCloud = generatePointCloud(left, smoothedMap, camDistance,
       focalLength);
export(pointCloudExportPath, pointCloud);
```

As described in the design section, the program loads the image and cuts it in a left and right part. These images are then passed in the `iterativeDisparityMap` function. This function requires an additional parameter, being the downscale factor of the base disparity map. More details about this are explained in the corresponding section.
This disparity map is then denoised and smoothed. It is given to the `generatePointCloud` function, which also needs the distance between the cameras d and the focal length f, as it is necessary to compute the depth. Finally, the point cloud is exported as a `.ply` file, to be viewed in MeshLab.

5.3.2. Computing the Disparity Map. To generate the disparity map, the algorithm searches for the most similar pixel on the other image. This is done by comparing the pixels with a cost function, and taking the pixel with the minimal cost.

As mentioned in the design section, there are constraints which limit the pixels that each pixels have to be compared with. Because the horizontal lines of the images represent the same line on the scene, it is not necessary to compare pixels at other rows. An upper bound for the disparity can also be set, since there is no object touching the camera.

```
int pixelDisparity(PImage left, PImage right, int x, int y,
int rangeMin, int rangeMax, int window) {
  int bestx = x, minCost = Integer.MAX_VALUE;
  for (int i = rangeMin; i < rangeMax; i++) {
    int c = cost(left, right, x, i, y, window);
    if (c < minCost) {
      minCost = c;
      bestx = i;
    }
  }
  return bestx - x;
}
```

The `rangeMin` and `rangeMax` set an upper and lower bound to the disparity. This limits the pixels that have to be searched through. When computing the disparity without base map, the range is 20% of the images width. The disparity of the pixel with the highest similarity is returned. This process is repeated for every pixel on the image to produce a disparity map.

5.3.2.1. **Cost Function**. A large variety of cost function exist. Some of which are compared in [9]. A basic and effective cost function is the sum of squared difference. As the name of this cost function implies, it compares pixels and returns the squared difference of the pixels colours. This difference is summed over all the pixels in the given window.

In this implementation of the cost function, `xleft` is the x coordinate of the pixel that we want to compare on the left image and `xright` the x coordinate of the pixel on the right image. The y coordinates of the pixel is always the same on the right and on the left.

```
int sum = 0;

for (int i = -window / 2; i <= window / 2; i++)
for (int j = - window / 2; j <= window / 2; j++) {
  sum += colorDistance(left.get(xleft + j, y + i), right.
     get(xright + j, y + i));
}

return sum;
```

The `colorDistance` function separates the red, green and blue channel, and squares the difference of the colours at the left and right pixels. Since this function is heavily used, it has optimised to reduce the computation time, in the detriment of the readability of the code.

```
int sum = 0, diff = 0;
//red
diff = (color1 >> 16 & 0xFF) - (color2 >> 16 & 0xFF);
sum += diff * diff;
//green
diff = (color1 >> 8 & 0xFF) - (color2 >> 8 & 0xFF);
sum += diff * diff;
//blue
diff = (color1 & 0xFF) - (color2 & 0xFF);
sum += diff * diff;
return sum;
```

Let's compare the disparity map produced using this algorithm. The following image will be used as input data:

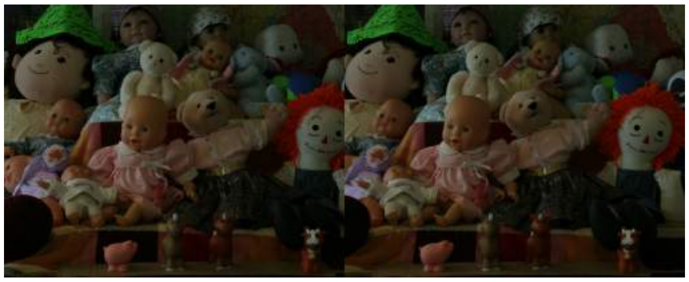


Fig. 8: Stereo image pair used to create the disparity map in the examples

To display the disparity map, an image is created where higher disparity is shown as darker. A way to view these images, is to consider the brightness of the pixel, the depth.

(a) Disparity map generated with window size of 2

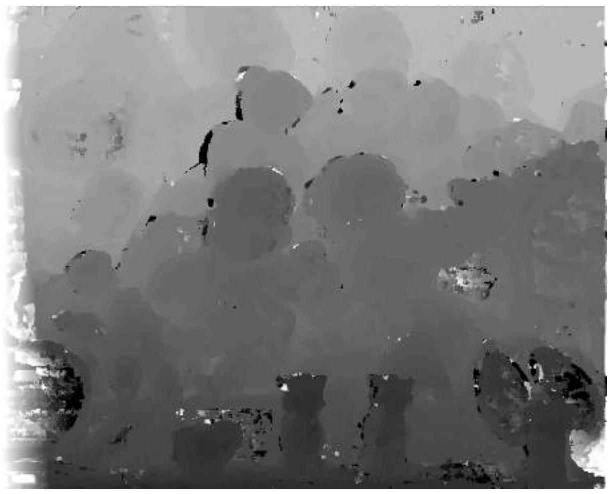

(b) Disparity map generated with window size of 7

Fig. 9: Disparity map generated using different window size for the cost function

Using a small window, results in a disparity map with a lot of noise, especially high frequency noise. The general depth however is well captured in the map. When a larger window is chosen, the noise decreases significantly, all though some low frequency noise remains. The compromise for the quality of the disparity map is the processing time. The window area is around 12 times bigger, thus the processing time grows by the same amount. In this important to find a good quality to time balance. The noise appearing in this process will have to be removed in denoising step.

5.3.2.2. **Disparity Map Generation using Base Disparity Map**. To accelerate the generation of the disparity map, this process is divided in two steps; generate a base disparity map from a smaller version of the image, and use it to generate the full size map.

First, the image currently being processed is scaled down. A disparity map of this scaled down version of the image is generated. This map will be called the base map from now on. The base map is first denoised and blurred, then the disparity map of the full size image created, using the base disparity map to reduce the searching range.

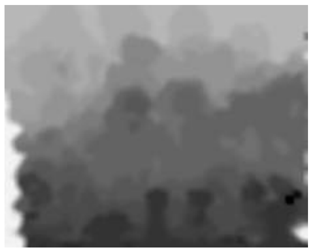
To use the base map when computing the disparity of a pixel, the program looks into the base map at the corresponding area, and sets this as the base disparity of this pixel. Then, the program looks left and right of that pixel, by twice the down scaling factor of the base map.

```
int baseDisparity = baseMapScale*baseMap[y / baseMapScale]
     [x / baseMapScale];
disparityMap[y][x] = pixelDisparity(left, right, x, y,
    x + baseDisparity - baseMapScale * 2,
    x + baseDisparity + baseMapScale * 2,
    window);
```

The advantage of this technique is that a lot less pixels have to be compared in total. For an image width 300 pixels width, the default range is 20% of the image width, so 60 pixels. However, using a base map a third the size, only 32 pixels would be compared. 20% of the 100 pixels width for the base image, so 20 pixels. And an additional $2 \times 2 \times baseMapScale = 12$ pixels to search through the possible pixels around the base disparity. Resulting in almost halving the number of comparison required. For larger images, the number of pixels to compare can decrease up to $1/baseMapScale$ of the default approach.

If the base map is much smaller than the original image, for example by a factor of 10, the disparity map produced will become very noise. A good balance is to have a base map a third the size of the original one.

A disadvantage of this algorithm is that if the base map contains errors, there will also be error in the disparity map. Usually, this will be low frequency noise since the errors will come in blocks.


(a) Base disparity map generated with standard stereo matching algorithm

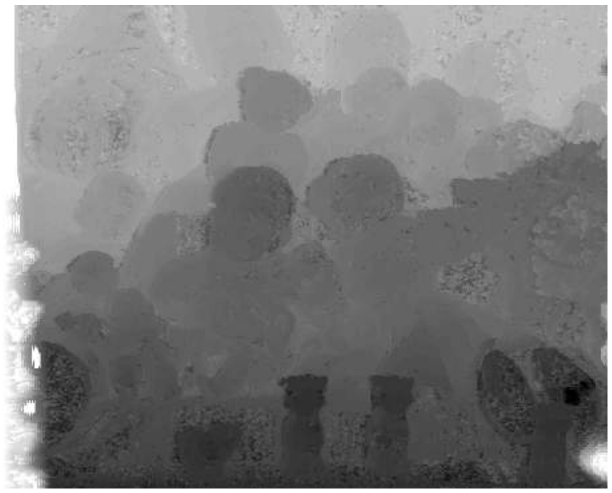

(b) Disparity map generated with the previous base map

Fig. 10: Disparity map of full scale image using the base disparity map

5.3.3. Denoising. As has been seen in the previous examples, noise appears often in the generated maps. To mitigate this, an algorithm to reduce the noise in the disparity maps has to be created.

The noise is divided in two categories, low and high frequency noise. To remove this noise, it first has to be found, then the noisy disparity is replaced with a more adequate disparity.

5.3.3.1. **Detecting High Frequency Noise**. This noise is distinguished by being single dots of a different disparity in an area where the area has the same disparity. The function to find these disparities looks at the neighbouring pixels, and checks that most of them have a similar disparity.

```
boolean highFreq(float[][] disparityMap, boolean low,
    int x, int y) {
  int count = 0;
  for (int i = y - 1; i <= y + 1; i++) {
    for (int j = x - 1; j <= x + 1; j++) {
      if (abs(disparityMap[i][j] - disparityMap[y][x]) ==
     0)
        count++;
      if (count == 5)return true;
    }
  }
  return false;
}
```

In this technical deliverable, the required of pixels having the same disparity to not be qualified as noise is 5. This may seem odd since this is more than half of the 8 neighbours, making shark angles at edges impossible. However, during the correction of the pixels, a the edge may take back its original disparity. This will be seen in the correction paragraph.

5.3.3.2. **Detecting Low Frequency Noise**. As has been explained in the design section, low frequency noise is detected, not by looking at the direct neighbours but at a circle around that point. These points are considered noise, if there is not a certain percentage of the similar disparities in that circle.

```
boolean lowFreq(float[][] disparityMap, int radius,
    int threshold, int x, int y) {
  float angle = 0;
  int counter = 0;
  while (angle < TWO_PI) {
    int k = x + (int)(cos(angle) * radius);
    int l = y + (int)(sin(angle) * radius);
    if (abs(disparityMap[y][x] - disparityMap[l][k]) <=
     threshold)
        counter++;
    if(counter >= 0.1*n)return true;
    angle += 1f/radius;
  }

  return false;
}
```

This method has been simplified to demonstrate the concept of the algorithm. In the implementation, checks have to be done to verify that the values `k` and `l` are not out of the range of the array.

5.3.3.3. **Correcting the Noise**. To correct the noise, the neighbour with the most similar colour to the pixels having a noisy disparity is searched. If this pixel is not noise, then the disparity of that pixel is assigned to the noisy pixel. In case that the neighbour with the most similar colour is noise, the second closest is taken. And in the case where no neighbour is valid, the disparity is unchanged.

When correcting the noise, it is not always possible to find a neighbour with that is not noise. However, by repeating this correction for several steps, the noise areas will shrink. Therefore, the denoising step is repeated a multiple times to reduce the noise as much as possible.

To visualise the robustness of this algorithm, a very noise image is given as input.

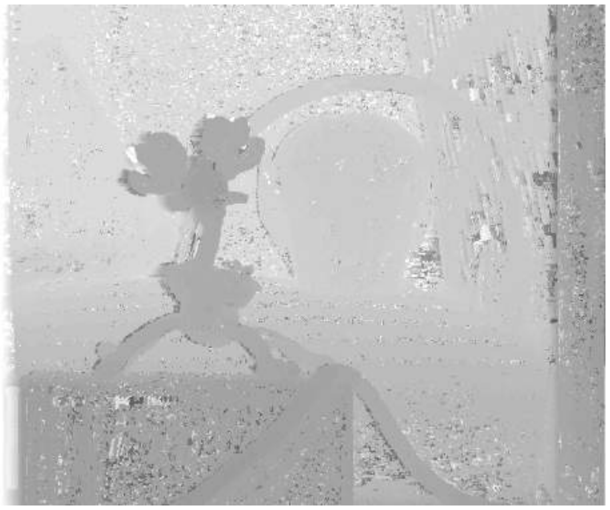


Fig. 11: Disparity map with large amount of noise

This disparity map suffers from a lot of high and low frequency noise. When denoising this map using the denoising algorithm, the following is produced:

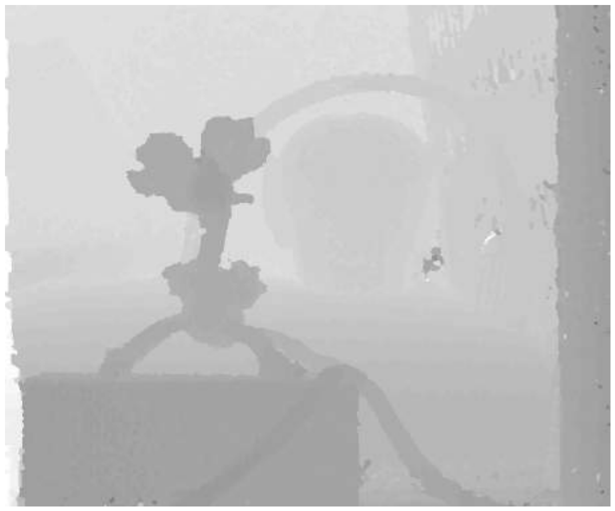


Fig. 12: Above disparity map passed through denoising algorithm

The downside of this method is that the edges of the objects become less smooth. Thin features may also disappear, as well as small details. However, since this is generating new information, it cannot have a 100% accuracy.

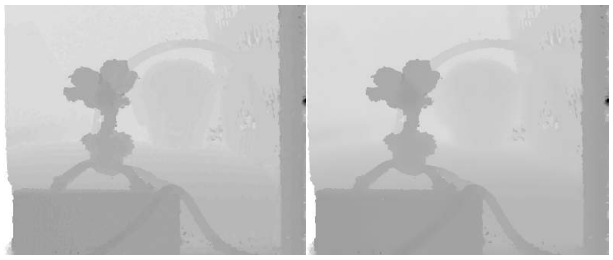
5.3.4. Smoothing the Disparity Map. Before this step, the disparity maps contain only integer value. This can be seen in forms of cuts between the different region of the disparity maps, even with very close disparities.

As has been described in the design section, the algorithm to smooth the image performs a blur over pixels with a similar disparity in a large region around that point.

```
float similarAverage(float[][] disparityMap, int x, int y,
    int window, int threshold) {
  float sum = 0;
  int n = 0;
  for (int i = y - window / 2; i <= y + window / 2; i++) {
    for(int j = x - window/2; j <=  x + window/2; j++) {
      if (abs(disparityMap[y][x] - disparityMap[i][j]) <=
    threshold) {
        sum += disparityMap[i][j];
        n++;
      }
    }
  }

  return sum / n;
}
```

This function returns the average of the similar disparities in a square around the point of interest. This value is searched for every pixel on the image and given as the new disparity.



(a) Disparity map generated without smoothing (b) Disparity map generated with smoothing

Fig. 13: Disparity map before and after smoothing

The advantage of this method is that the edges are kept, whilst the wrong edges, meaning edges where there should be floating point values, are smoothed out. The worth of this step can be seen more easily when looking at the point cloud. Without the smoothing, the point cloud is divided in different layers. Smoothing the map makes the point cloud look more organic.

5.3.5. Generating the Point Cloud from the Disparity Map. The mathematics of the process to compute the depth and project the pixels from an 2D image back to a 3D volume have been derived in the design section.

```
Point pointFromDisparity(PImage image, float disparity, int
      x, int y, float lensDistance, float focalLength) {
  //normalize position to center
  float nx = 2 * (float)x / image.width  - 1;
  float ny = 2 * (float)y / image.height - 1;
  //find tangent of angle
  float xTan = nx / focalLength;
  float yTan = ny / focalLength * (image.height / image.
    width);
  //compute depth
  float depth = - focalLength * lensDistance * image.width
    / disparity - focalLength;
  return new Point(
    xTan * (depth + focalLength),
    yTan * (depth + focalLength),
    depth,
    image.get(x, y));
}
```

A point is computed for each pixel on the image. A `ply` file is the generated, containing a list of points with a colour. Here are a few point clouds generated using this program.

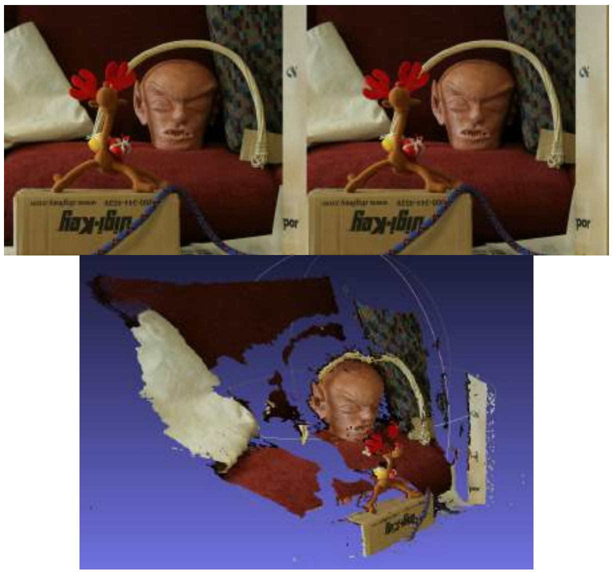


Fig. 14: Stereo image pair (top) and point cloud generated with the algorithm (bottom)

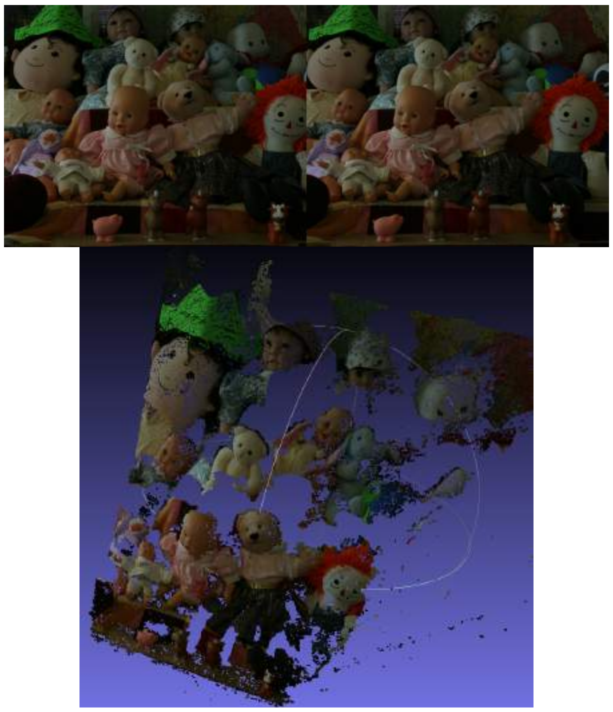


Fig. 15: Stereo image pair (top) and point cloud generated with the algorithm (bottom)

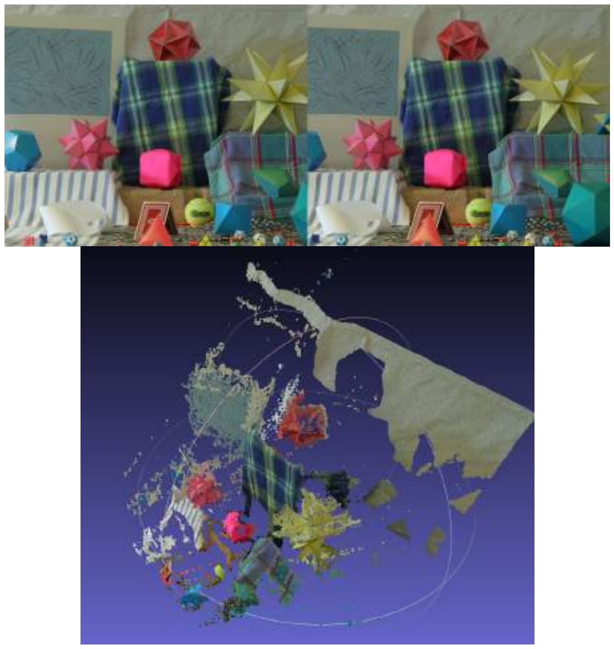


Fig. 16: Stereo image pair (top) and point cloud generated with the algorithm (bottom)

5.4. Assessment

The program produced in this technical deliverable is able to generate point clouds of stereo pairs and passes the requirements set. To develop this program, a number of iterations have been done, and a lot of other approaches were tried unsuccessfully.

Some unsuccessful approaches could however be improved, and used more successfully. For example, using the edges to have a sharper contour of the shapes would be beneficial. Currently, the disparity bleeds to neighbouring objects, which can be seen in the generated point clouds. Using edges could mitigate this issue.

Some other constraints, like each pixel only matching to a single other pixel, have not yet been implemented and verified. This could be done using monotony. Performance optimisations could also be looked at, during the matching process and the low frequency noise removal. Implementing parallelism in the program would also improve the performance.

A larger addition would be implementing point cloud registration; a set of stereo pair images are taken, and the point clouds generated from the single stereo images are merged together to form a higher quality and denser point cloud.

Acknowledgement

The author would like to show gratitude towards the entire BiCS class, which helped the author to stay motivated. The author also wishes to thank the tutor for the support throughout the development of this bachelor project.

6. Conclusion

In this report, two deliverables were made that satisfied their requirements.

In the scientific deliverable, four different techniques were presented, that used visual inputs to generate three dimensional objects. Each method used a different approach, all having their advantages and disadvantages. This variety of algorithm came from the equally vast possible requirements.

In the technical deliverable, a program was made, that was able to generate point clouds from a pair of stereo images. Though the quality of the point cloud is not excellent and comparable to state of the art technique, approaches for solving this kind of computer vision problem were learnt and the the results remain acceptable.

The author wishes to express that this producing this semester project taught him the range of possibilities in computer vision, but that it is likely applicable in other fields. Incidentally, the author looks forward to witness the continuous development of computer vision.

7. Plagiarism statement

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgement amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
    1) Not putting quotation marks around a quote from another person's work
    2) Pretending to paraphrase while in fact quoting
    3) Citing incorrectly or incompletely
    4) Failing to cite the source of a quoted or paraphrased work
    5) Copying/reproducing sections of another person's work without acknowledging the source
    6) Paraphrasing another person's work without acknowledging the source
    7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
    8) Using another person's unpublished work without attribution and permission ('stealing')
    9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[BiCS(2021)] BiCS Bachelor Semester Project Report Template. https://github.com/nicolasguelfi/lu.uni.course.bics.global University of Luxembourg, BiCS - Bachelor in Computer Science (2021).

[BiCS(2021)] Bachelor in Computer Science: BiCS Semester Projects Reference Document. Technical report, University of Luxembourg (2021)

[1] João Carreira,Fuxin Li,Cristian Sminchisescu. Object Recognition by Sequential Figure-Ground Ranking DOI 10.1007/s11263-011-0507-2, Springer Science+Business Media, LLC 2011

[2] Oxford Learner's Dictionaries: Parallax. www.oxfordlearnersdictionaries.com

[3] Emmanuel Prados, Olivier Faugeras. Shape from Shading

[4] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards Robust Monocular Depth Estimation:Mixing Datasets for Zero-shot Cross-dataset Transfer IEEE Transactions On Pattern Analysis And Machine Intelligence, 2020

[5] A.D. Jepson and D.J. Fleet. 2503. Epipolar Geometry University Of Toronto, CSC420, Introduction to Image Understanding

[6] Se, Stephen and Piotr Jasiobedzki.. "Stereo-Vision Based 3D Modeling and Localization for Unmanned Vehicles." International Journal of Intelligent Control and Systems, VOL. 13, NO. 1, MARCH 2008, 46-57

[7] Carsten Griwodz, Simone Gasparini, Lilian Calvet, Pierre Gurdjos, Fabien Castan, et al.. AliceVision. Meshroom: An open-source 3D reconstruction pipeline. 12th ACM Multimedia Systems Conference (MMSys 2021), Sep 2021, Istanbul, Turkey. pp.241-247, ff10.1145/3458305.3478443ff. ffhal-03351139

[8] Jiun-Hung Chen. Lecture 6: Features and Image Matching University of Washington, CSE 455, Computer Vision, Winter 2009

[9] H. Hirschmüller and D. Scharstein. Evaluation of cost functions for stereo matching. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, MN, June 2007.

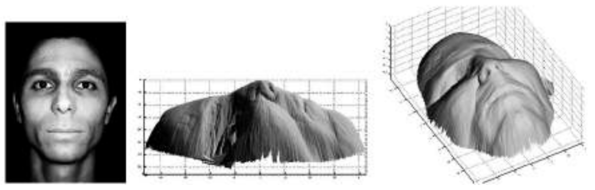
[10] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1/2/3):7-42, April-June 2002.

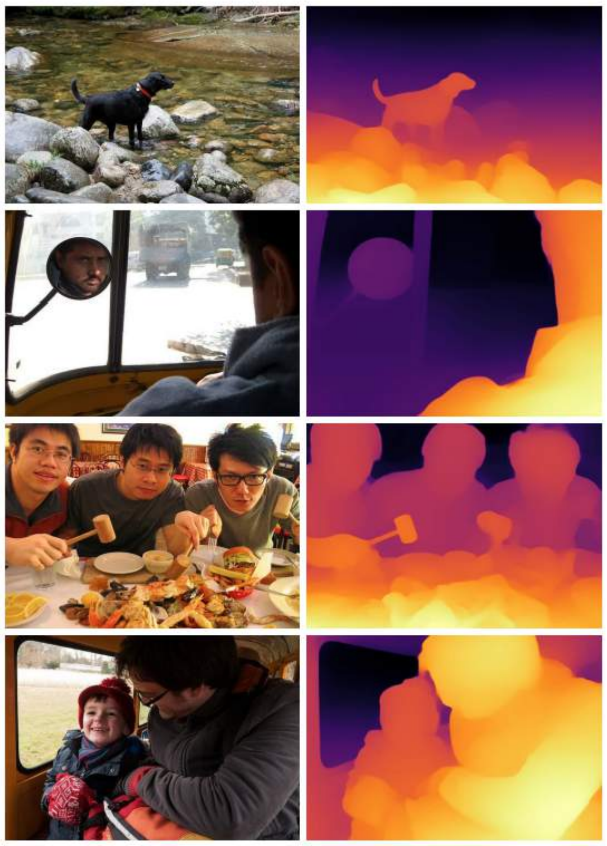
[11] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), volume 1, pages 195-202, Madison, WI, June 2003.

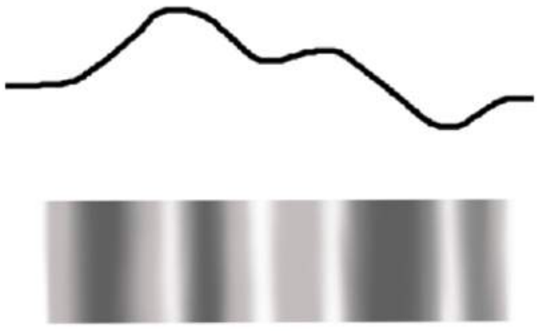
[12] D. Scharstein and C. Pal. Learning conditional random fields for stereo. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, MN, June 2007.

[13] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nesic, X. Wang, and P. Westling. High-resolution stereo datasets with subpixel-accurate ground truth. In German Conference on Pattern Recognition (GCPR 2014), Münster, Germany, September 2014.

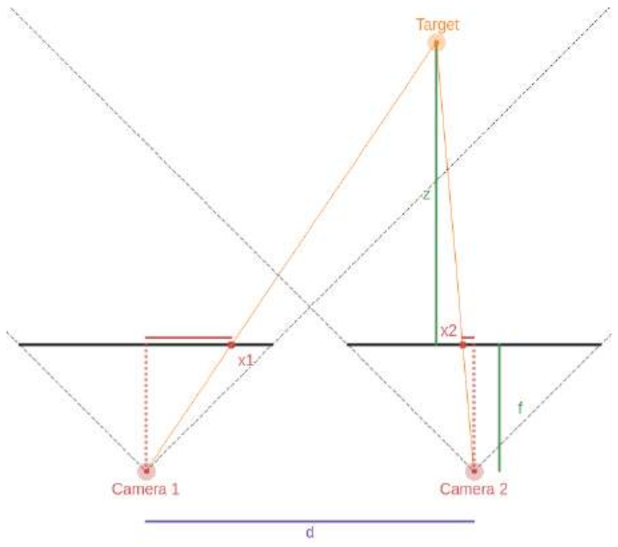
8. Appendix


Gray scale image of a face as visual input, and three dimensional shape generated with an algorithm described in the paper

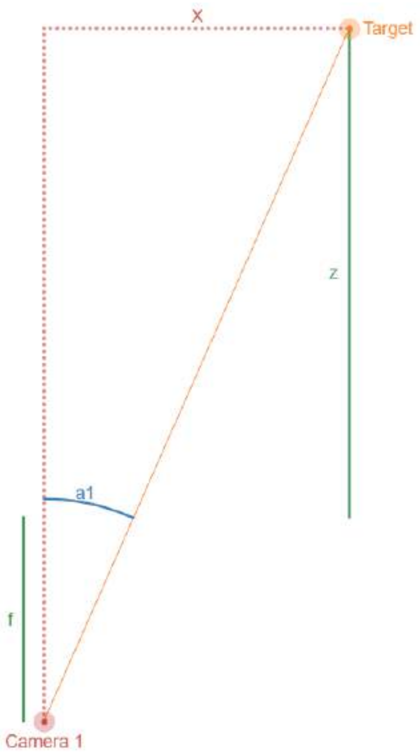

Left: input image, Right: generated depth map with model in paper


Top: Shape of the surface, Bottom: Shade of the surface

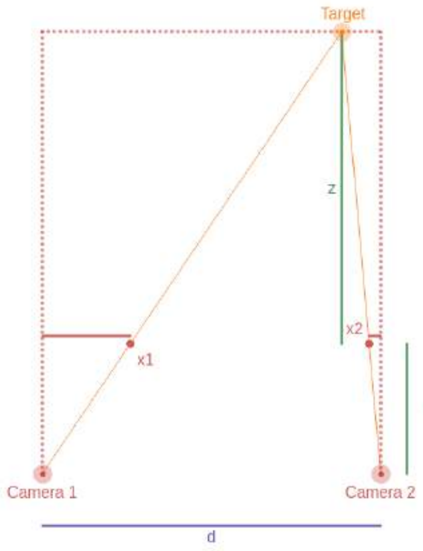
Scene recreated using hand held camera with method proposed in [6]



Two camera with distance 2 between each other. x1 and x2 lay on the image plan and are where the target is displayed on the image. f is the focal length and z the depth of the target.



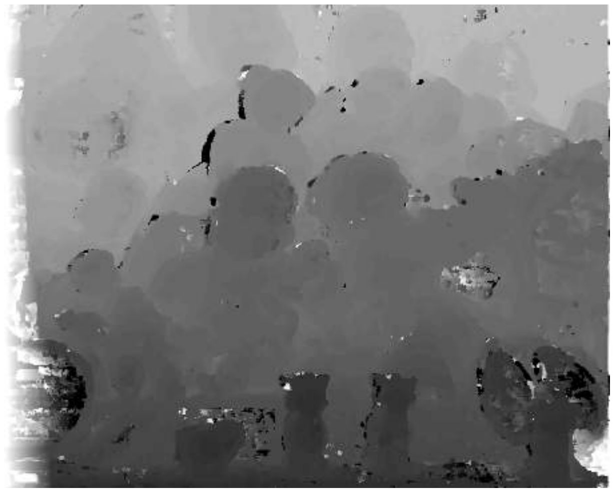
Right triangle. a1 is angle to the target pixel. X horizontal distance of the target point with the camera. f is the focal length and Z is the depth between the target point and image place.



Two right triangles formed with the two cameras, the target, and the points at the horizontal position of the cameras and depth of the target. x1 is the horizontal position of the target on the image plane of camera 1, and x2 is the same but on camera 2. d is the distance between the camera, z is the depth of the target, and f the focal length.

Stereo image pair used to create the disparity map in the examples

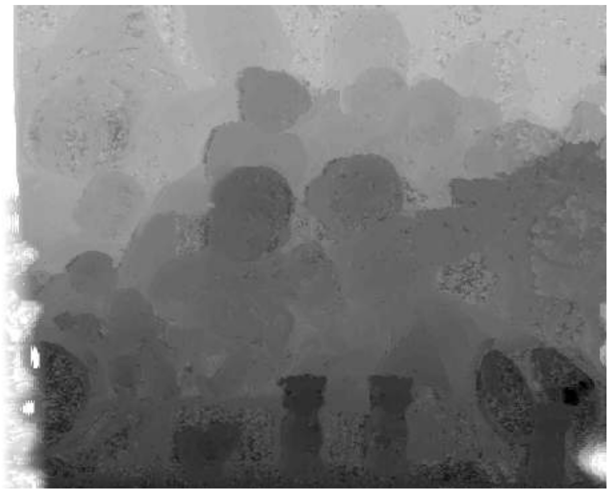
(a) Disparity map generated with window size of 2



(b) Disparity map generated with window size of 7

Disparity map generated using different window size for the cost function

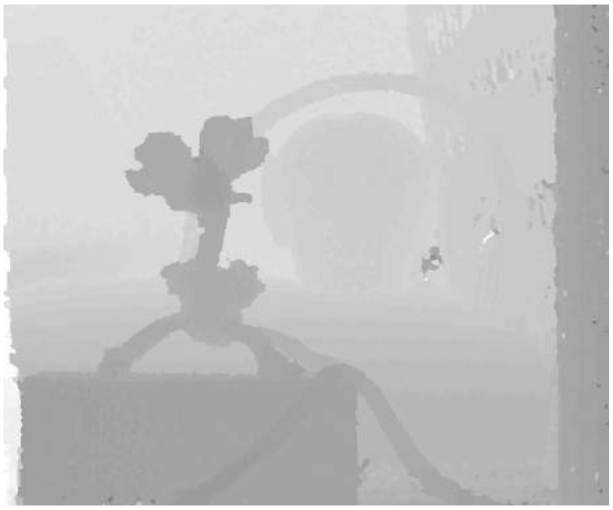
(c) Base disparity map generated with standard stereo matching algorithm



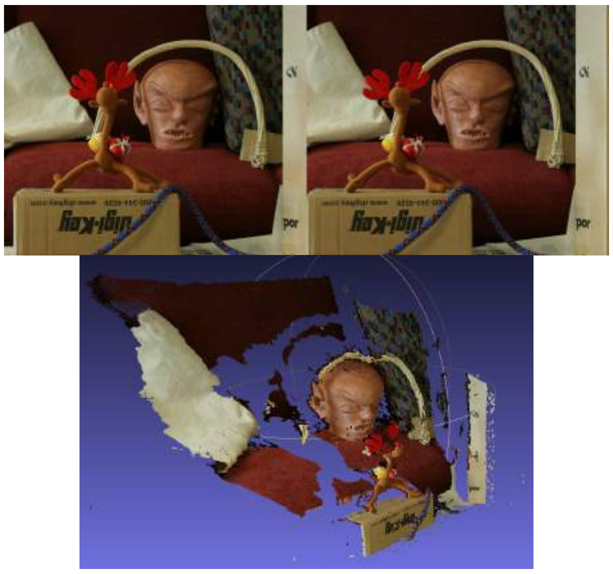
(d) Disparity map generated with the previous base map

Disparity map of full scale image using the base disparity map

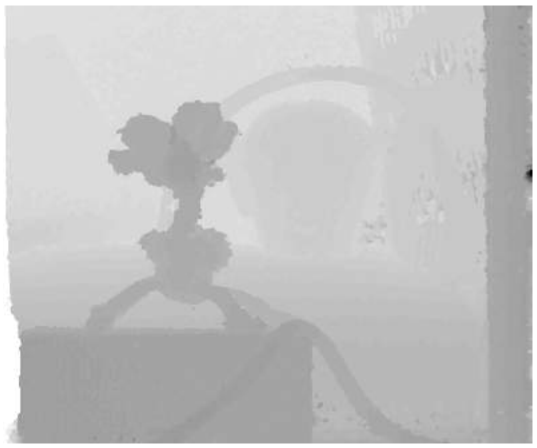
Disparity map with large amount of noise



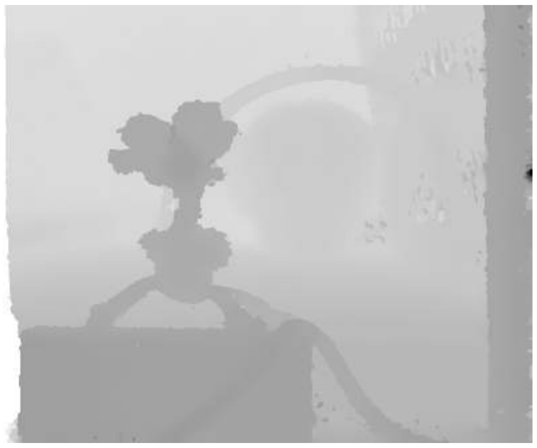
Above disparity map passed through denoising algorithm



Stereo image pair (top) and point cloud generated with the algorithm (bottom)

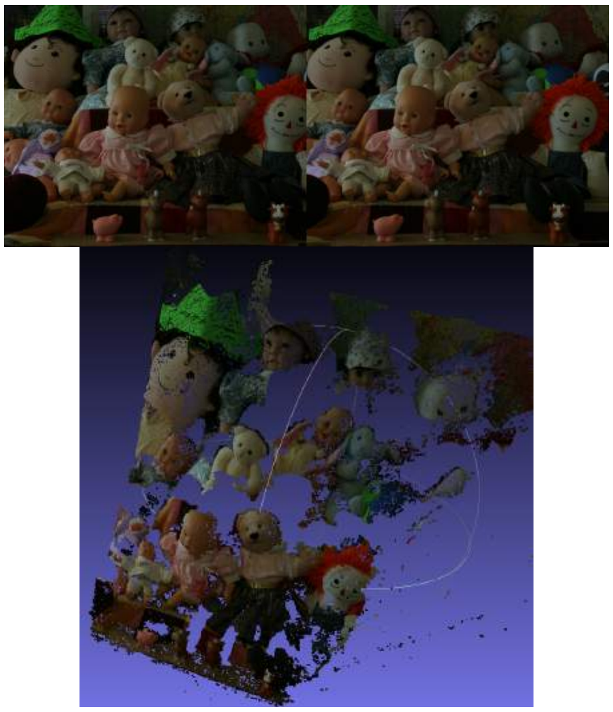


(e) Disparity map generated without smoothing

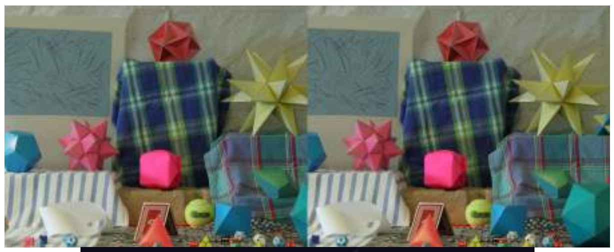
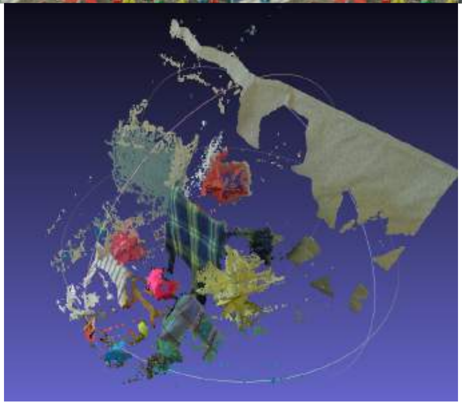


(f) Disparity map generated with smoothing

Disparity map before and after smoothing



Stereo image pair (top) and point cloud generated with the algorithm (bottom)



Stereo image pair (top) and point cloud generated with the algorithm (bottom)