Stereo: from pixels to 3D surface mesh

3D Computer Vision

Ferdi van der Heijden ▸ University of twente - RAM ▸ 10/1/2016

Contents

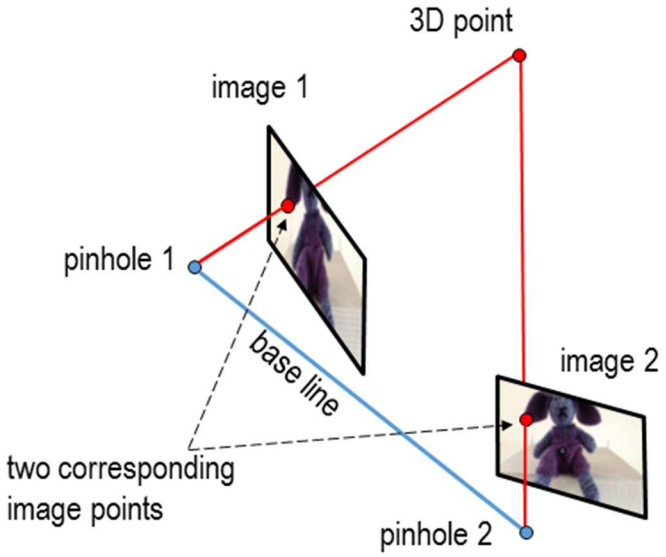
Stereo: from pixels to 3D surface mesh

3D Computer Vision

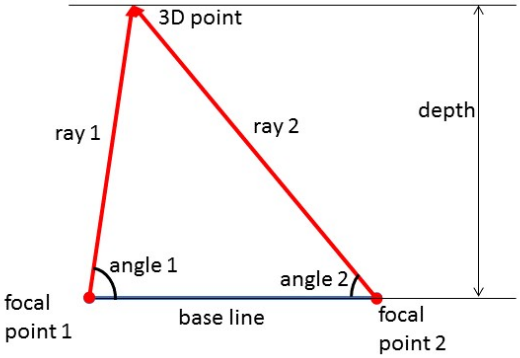
1 Introduction

The picture above shows a *stereo anaglyph*. It provides the illusion of depth when viewed through color-coded anaglyph glasses. Each of the two glasses should have its own chromatic filter; typical red and cyan.

The principle of estimating depth from images taken from two different points of view is *triangulation*. A triangle is formed between the two focal points of the two cameras and the 3D point of interest.



Knowledge of the length of the *base line* (the line between the two focal points), and the two angles formed by the base line and the two rays from the 3D point, suffices to calculate the position of the 3D point in 3D space.



In case of a non-cross-eyed set-up of the cameras, the difference between the two angles is encoded in the *disparity* of the imaged 3D point. Disparity is the spatial shift of the point in the two images. The disparity is seen in the stereo anaglyph as the horizontal shifts between red and cyan copies of imaged objects.

The mathematics of stereo vision relies on the concept of projective spaces. The situation with two cameras is a sub-branch of that which is known as *epipolar geometry*. We will introduce this in the next section.

We assume that the cameras are fully calibrated. This implies that the pose of one camera relative to the other camera is precisely known. These cameras will never be exactly aligned, so that we must assume that the system is cross-eyed. The situation is much simpler if they would be perfectly aligned without any squint. Therefore, the first step to extract depth information is to virtually rotate the cameras by applying projective transforms to the images such that it looks as if the cameras are fully aligned. This process is called *stereo rectification*. It facilitates finding the disparities of the images. The estimation of the disparities is called *stereo matching*. Different algorithms exist for doing this. Some of them will be discussed in a subsequent section.

With known disparities, the 3D reconstruction of the surface is easily done. The result is a *dense depth map*. For each pixel in one of the two images, the depth is specified. Creating a 3D point cloud and 3D surface mesh will also be easy then. This will be the end point of this chapter.

In practice, however, some complications often occur. One complication is the occurrence of *occlusions*: a 3D surface patch is visible in one image, but is hidden (occluded) in the other image. If detected, such an occlusion gives rise to a hole in the surface mesh. If not detected, it will give rise to an erroneous surface patch.

Similar problems arise in case of textureless surfaces. If a wall is completely white, for instance, stereo matching is likely to fail. When detected, it gives rise to holes in the mesh; when not detected, the mesh will be very inaccurate. Repairing holes or errors in the mesh is not discussed in this chapter.

Another complication occurs with a multiple camera system. How to combine the information from 3 or more cameras? If a surface patch is visible in more than 2 cameras then combining the information could improve the accuracy of the mesh. One strategy would be to create a 3D surface mesh from each pair of cameras. These surfaces could be stitched to create a larger range of surfaces.

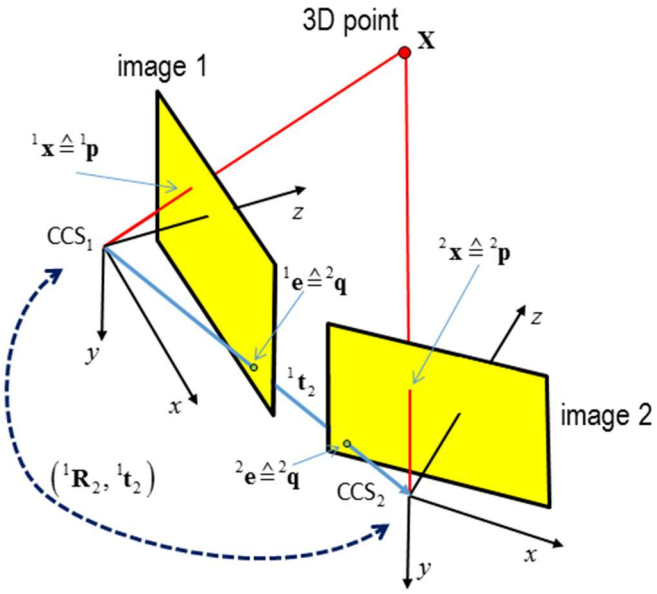
One last remark here is that opposite to *dense stereo* we have *sparse stereo*. This is stereo vision applied to only a few selected interest points (synonyms: key points, visible landmarks). This topic will be addressed in another chapter.

2 Epipolar geometry

This section is an introduction to epipolar geometry. This geometry describes the relations between a 3D point and its projected locations in the image planes of two cameras.

2.1 The geometry of two cameras

The figure below shows a 3D overview of the set-up of a stereo camera.



A 3D point $\mathbf{X}$ is imaged by two camera systems. Each camera has its own coordinate system: CCS_1 and CCS_2. The 3D point can be represented in either of the two CS. Notation: ${}^1\mathbf{X}$ and ${}^2\mathbf{X}$. The two coordinate systems are related by means of a translation and a rotation. The translation is the shift of the origins. We have already introduced this as the *base line*. Expressed in CCS_1, the base line vector is denoted by ${}^1\mathbf{t}_2$, i.e. the origin of CCS_2. The orientation of CCS_2 relative to CCS_1 is defined by the rotation matrix ${}^1\mathbf{R}_2$. With that, any point that is represented in CCS_2 is represented in CCS_1 by:

$$ {}^1\mathbf{X} = {}^1\mathbf{R}_2\,{}^2\mathbf{X} + {}^1\mathbf{t}_2 \tag{1} $$

In homogeneous 4D coordinates, this becomes:

$$ {}^1\underline{\mathbf{X}} = {}^1\mathbf{T}_2\,{}^2\underline{\mathbf{X}} \quad \text{with} \quad {}^1\mathbf{T}_2 = \left[\begin{array}{ccc|c} & {}^1\mathbf{R}_2 & & {}^1\mathbf{t}_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2} $$

The reverse relations are given by:

$$ \begin{aligned} {}^2\mathbf{X} &= {}^1\mathbf{R}_2^T\left({}^1\mathbf{X} - {}^1\mathbf{t}_2\right) \\ &= {}^2\mathbf{R}_1\,{}^1\mathbf{X} + {}^2\mathbf{t}_1 \\ {}^2\underline{\mathbf{X}} &= {}^2\mathbf{T}_1\,{}^1\underline{\mathbf{X}} \end{aligned} \qquad \text{with:} \qquad \begin{aligned} {}^2\mathbf{R}_1 &= {}^1\mathbf{R}_2^{\mathrm{T}} \\ {}^2\mathbf{t}_1 &= -{}^2\mathbf{R}_1\,{}^1\mathbf{t}_2 \\ {}^2\mathbf{T}_1 &= {}^1\mathbf{T}_2^{-1} \end{aligned} \tag{3} $$

We assume that the two cameras are fully calibrated, and that nonlinear lens distortions are either negligible or compensated. The camera calibration matrices are $\mathbf{K}_1$ and $\mathbf{K}_2$. The focal distances are d_1 and d_2.

The image of a 3D point

The 3D point $\mathbf{X}$ is imaged in the first and in the second image. Regarded as 3D points in the image planes the coordinates of these image points, represented in CCS_1 and CCS_2, follows from the formulas of perspective projection:

$$ {}^1\mathbf{x} = \frac{1}{Z_1}\begin{bmatrix} X_1 \\ Y_1 \\ d_1 Z_1 \end{bmatrix} = \frac{d_1}{Z_1}\,{}^1\mathbf{X} \qquad {}^2\mathbf{x} = \frac{1}{Z_2}\begin{bmatrix} X_2 \\ Y_2 \\ d_2 Z_2 \end{bmatrix} = \frac{d_2}{Z_2}\,{}^2\mathbf{X} \tag{4} $$

where X_1, Y_1 and Z_1 are the coordinates of ${}^1\mathbf{X}$, and X_2, Y_2, Z_2 likewise of ${}^2\mathbf{X}$. Expressed in 2D pixel coordinates, the homogeneous representation becomes:

$$ \begin{aligned} {}^1\underline{\mathbf{p}} &= \mathbf{K}_1\left[\mathbf{I}_{3\times3} \quad \mathbf{0}_{3\times1}\right]{}^1\underline{\mathbf{X}} = \mathbf{K}_1\,{}^1\mathbf{X} \\ {}^2\underline{\mathbf{p}} &= \mathbf{K}_2\left[\mathbf{I}_{3\times3} \quad \mathbf{0}_{3\times1}\right]{}^2\underline{\mathbf{X}} = \mathbf{K}_2\,{}^2\mathbf{X} \end{aligned} \tag{5} $$

Note that ${}^1\mathbf{x}$ and ${}^1\mathbf{p}$ refers to the same point. ${}^1\mathbf{x}$ is the 3D point represented in CCS_1, whereas ${}^1\mathbf{p}$ is the 2D point represented in pixel coordinates.

The epipoles

The epipole ${}^1\mathbf{e}$ is the mapping of the focal point of the second camera in the image plane of the first camera. Thus, according to (4), the 3D representation of the epipole ${}^1\mathbf{e}$ is:

$$ {}^1\mathbf{e} = \frac{d_1}{{}^1t_z}\,{}^1\mathbf{t}_2 \tag{6} $$

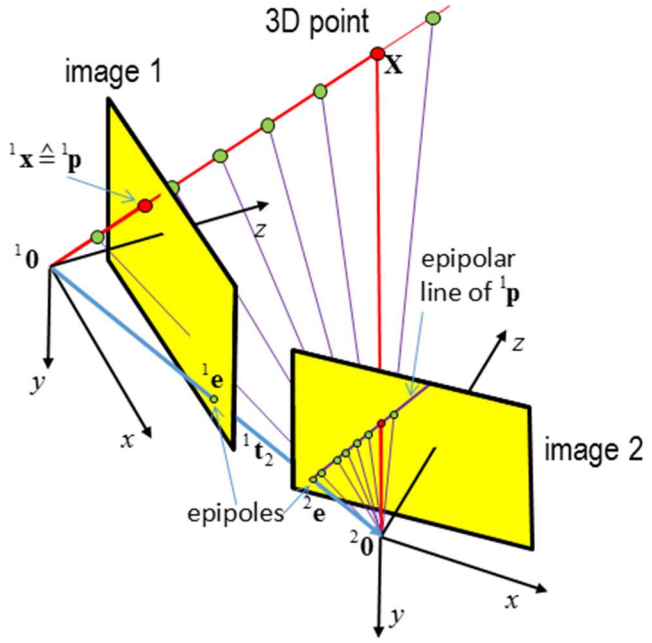
in which ${}^1t_z$ is the z-coordinate of the vector ${}^1\mathbf{t}_2$. Regarded as a 2D point in the image, and expressed in pixel coordinates, we denote the epipole, by ${}^1\mathbf{q}$. The epipole follows then from (5):

$$ {}^1\underline{\mathbf{q}} = \mathbf{K}_1 {}^1\mathbf{t}_2 \tag{7}$$

Likewise expressions hold for ${}^2\mathbf{e} \triangleq {}^2\mathbf{q}$ being the epipole in the second camera, i.e. the focal point of the first camera mapped on the image plane of the second camera.

2.2 The essential matrix and the fundamental matrix

In the figure below, the focal points are denoted by ${}^1\mathbf{0}$ and ${}^2\mathbf{0}$, respectively. The points $\left(\mathbf{X}, {}^1\mathbf{0}, {}^2\mathbf{0}\right)$ form a triangle which defines a 2D plane in the 3D space. The two vectors ${}^1\mathbf{X}$ and ${}^1\mathbf{t}_2$ span this plane. The vector ${}^2\mathbf{X} \triangleq {}^1\mathbf{X} - {}^1\mathbf{t}_2$ is also lying in this plane. Therefore, the vectors ${}^1\mathbf{X}$, ${}^1\mathbf{t}_2$, and ${}^1\mathbf{X} - {}^1\mathbf{t}_2$ must be coplanar.



The essential matrix

The cross vector product ${}^1\mathbf{t}_2 \times {}^1\mathbf{X}$ is a vector that is orthogonal to the plane. Consequently, it is also orthogonal to ${}^1\mathbf{X} - {}^1\mathbf{t}_2$. As a result, the co-planarity condition is expressed by:

$$ \left({}^1\mathbf{X} - {}^1\mathbf{t}_2\right)^{\mathrm{T}} \left({}^1\mathbf{t}_2 \times {}^1\mathbf{X}\right) = 0 \tag{8}$$

Substitution of (3) yields:

$$ \left({}^1\mathbf{R}_2 {}^2\mathbf{X}\right)^{\mathrm{T}} \left({}^1\mathbf{t}_2 \times {}^1\mathbf{X}\right) = 0 \tag{9}$$

With ${}^1\mathbf{t}_2 = \left[{}^1t_x \quad {}^1t_y \quad {}^1t_z\right]^{\mathrm{T}}$, the cross product operator ${}^1\mathbf{t}_2 \times$, symbolically written as $[{}^1\mathbf{t}_2]_\times$, can be written as a matrix multiplication with the antisymmetric matrix:

$$ [{}^1\mathbf{t}_2]_\times = \mathbf{T} = \begin{bmatrix} 0 & -{}^1t_z & {}^1t_y \\ {}^1t_z & 0 & -{}^1t_x \\ -{}^1t_y & {}^1t_x & 0 \end{bmatrix} \tag{10}$$

The co-planarity condition becomes:

$$ \left({}^1\mathbf{R}_2 {}^2\mathbf{X}\right)^{\mathrm{T}} \left(\mathbf{T}\, {}^1\mathbf{X}\right) = 0 \qquad \text{or:} \qquad {}^2\mathbf{X}^{\mathrm{T}} \left({}^2\mathbf{R}_1 \mathbf{T}\right) {}^1\mathbf{X} = 0 \tag{11}$$

Defining the 3×3 *essential matrix* as[1]

$$ \mathbf{E} \overset{def}{=} {}^2\mathbf{R}_1 \mathbf{T} \tag{12}$$

the condition simplifies to:

$$ {}^2\mathbf{X}^T \mathbf{E}\, {}^1\mathbf{X} = 0 \tag{13}$$

The ray from ${}^1\mathbf{0}$ to ${}^1\mathbf{X}$ is mathematically represented by $\gamma_1 {}^1\mathbf{X}$ with $\gamma_1 \in \mathbb{R}$. Likewise the ray from ${}^2\mathbf{0}$ to ${}^2\mathbf{X}$ is represented by $\gamma_2 {}^2\mathbf{X}$. Consequently, equation (13) holds for any pair of points on the two rays. Particularly, it holds for the two 3D points in the image planes:

$$ {}^2\mathbf{x}^T \mathbf{E}\, {}^1\mathbf{x} = 0 \tag{14}$$

This condition is called the *epipolar constraint*. The connotation is that if one 3D point in an image plane is fixed, e.g. ${}^1\mathbf{x}$, then the 2D plane of the triangle in the 3D space is fully defined. This plane is called the *epipolar plane* of $\mathbf{X}$. The position of the other 3D point, ${}^2\mathbf{x}$, cannot be chosen freely since it must be lying in this epipolar plane. The other constraint is that it must also lie in the image plane $z_2 = d_2$. Therefore, the degrees of freedom of ${}^2\mathbf{x}$ is just one: it is situated at the intersection line of the two planes. This intersection line is called the *epipolar line* of $\mathbf{X}$.

The rank of $\mathbf{E}$ is 2. This follows readily from the fact that if $\mathbf{X}$ is located on the base line, e.g. ${}^1\mathbf{X} = {}^1\mathbf{t}_2$, the triangle is

[1] The essential matrix can also be expressed in an alternative form. From (3), we have ${}^1\mathbf{t}_2 = -{}^1\mathbf{R}_2 {}^2\mathbf{t}_1$. Substitution in (9) yields: ${}^2\mathbf{X}^T {}^2\mathbf{R}_1 \left({}^1\mathbf{R}_2 {}^2\mathbf{t}_1 \times {}^1\mathbf{X}\right) = 0$ from which the alternative form $\mathbf{E} = [{}^2\mathbf{t}_1]_\times {}^2\mathbf{R}_1$ can be derived.

degenerated to a 3D line without defining a 2D plane. Mathematically: $\mathbf{E}\,{}^1\mathbf{t}_2 = {}^2\mathbf{R}_1\,{}^1\mathbf{t}_2 \times {}^1\mathbf{t}_2 = \mathbf{0}$. Thus, the vector ${}^1\mathbf{t}_2$ spans the null space of $\mathbf{E}$.

Since the epipole ${}^1\mathbf{e}$ has the same direction as ${}^1\mathbf{t}_2$, it must also be in the null space. We have the following property:

$$\mathbf{E}\,{}^1\mathbf{e} = \mathbf{E}\,{}^1\mathbf{t}_2 = \mathbf{0} \tag{15}$$

In conclusion: with given essential matrix $\mathbf{E}$ the direction of the base line vector and the epipole can be found by calculating the null space of $\mathbf{E}$, which can be done with singular value decomposition.

The fundamental matrix

Equation (14) expresses the epipolar constraint fully in the 3D space. There is also a version in terms of pixel coordinates. For that, we use (5) in the inversed form; that is ${}^1\mathbf{X} = \mathbf{K}_1^{-1}\,{}^1\underline{\mathbf{p}}$ and ${}^2\mathbf{X} = \mathbf{K}_2^{-1}\,{}^2\underline{\mathbf{p}}$. Substitution in (13) yields:

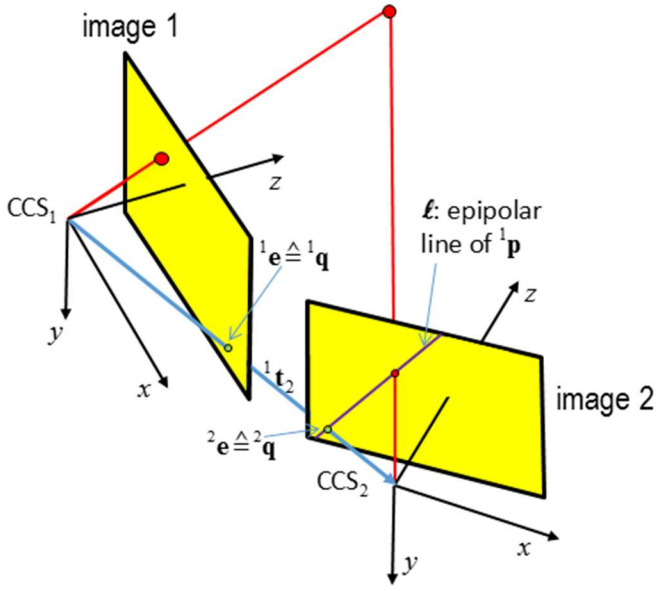
$$\begin{aligned}\left(\mathbf{K}_2^{-1}\,{}^2\underline{\mathbf{p}}\right)^{\mathrm{T}}\mathbf{E}\left(\mathbf{K}_1^{-1}\,{}^1\underline{\mathbf{p}}\right) &= 0\\ {}^2\underline{\mathbf{p}}^{\mathrm{T}}\left(\mathbf{K}_2^{-\mathrm{T}}\,\mathbf{E}\,\mathbf{K}_1^{-1}\right){}^1\underline{\mathbf{p}} &= 0\end{aligned} \tag{16}$$

By introduction of the *fundamental matrix*:

$$\mathbf{F} \overset{def}{=} \mathbf{K}_2^{-\mathrm{T}}\,\mathbf{E}\,\mathbf{K}_1^{-1} \tag{17}$$

the epipolar constraint takes the form:

$${}^2\underline{\mathbf{p}}^{\mathrm{T}}\,\mathbf{F}\,{}^1\underline{\mathbf{p}} = 0 \tag{18}$$



This result has the following connotation (see figure above). Suppose that we have selected a pixel of interest in the first image, so that the pixel coordinates ${}^1\underline{\mathbf{p}}$ is fixed. Then the vector $\boldsymbol{\ell} = \mathbf{F}\,{}^1\underline{\mathbf{p}}$ is orthogonal to ${}^2\underline{\mathbf{p}}$ as ${}^2\underline{\mathbf{p}}^T\boldsymbol{\ell} = 0$. The vector $\boldsymbol{\ell}$ can be regarded as the homogeneous representation of a line in the second image on which the point ${}^2\underline{\mathbf{p}}$ is lying. This line must be the *epipolar line* represented in pixel coordinates.

As an example, the figure below shows a stereo image pair, shown as an anaglyph. Image 1 is shown in cyan. The blue points in this image are selected more or less randomly. The corresponding points in image 2 are shown in red. The epipolar lines corresponding to the blue points are also displayed in red. It can be seen that all red points are all lying on the epipolar lines.

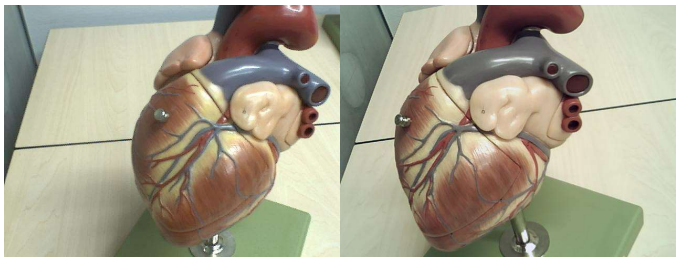
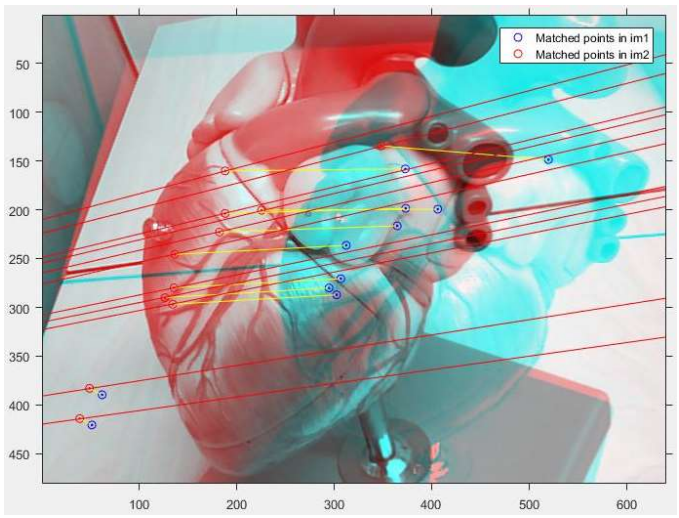


image 1 image 2



The null space of the fundamental matrix $\mathbf{F}$ follows from (15). Suppose that ${}^1\underline{\mathbf{q}} = \mathbf{K}_1\,{}^1\mathbf{e}$ is the epipole of the first image, expressed in homogeneous pixel coordinates, i.e. a 2D point in the image. Then:

$$\mathbf{F}\,{}^1\underline{\mathbf{q}} = \mathbf{0} \;\text{ since }\; \mathbf{F}\,{}^1\underline{\mathbf{q}} = \mathbf{F}\mathbf{K}_1\,{}^1\mathbf{e} = \mathbf{K}_2^{-\mathrm{T}}\,\mathbf{E}\,\mathbf{K}_1^{-1}\mathbf{K}_1\,{}^1\mathbf{e} = \mathbf{K}_2^{-\mathrm{T}}\,\mathbf{E}\,{}^1\mathbf{e} = \mathbf{0}.$$

Hence, the epipole ${}^1\underline{\mathbf{q}}$, expressed in homogeneous pixel coordinates, spans the null space $\mathbf{F}$. Consequently, if we

have the fundamental matrix, we can find the epipole in pixel coordinates by calculating the null space using singular value decomposition. By de-homogenizing the result, pixel coordinates in non-homogeneous coordinates are obtained.

In conclusion

Wrapping up the most important results, we have:

- The geometrical set-up of two camera is defined by the pose, ${}^1\mathbf{R}_2$, ${}^1\mathbf{t}_2$, and the two camera calibration matrices $\mathbf{K}_1$ and $\mathbf{K}_2$.
- The base line, defined by ${}^1\mathbf{t}_2$, intersects the two image planes at 3D positions ${}^1\mathbf{e}$ and ${}^2\mathbf{e}$. Expressed in pixel coordinates these points are denoted by ${}^1\underline{\mathbf{q}}$ and ${}^2\underline{\mathbf{q}}$ (homogeneous coordinates). These are the epipoles.
- The base line, together with a 3D point $\mathbf{X}$ define a triangle, and thus a 2D plane: the epipolar plane.
- If it is given that two points, ${}^1\mathbf{x}$ and ${}^2\mathbf{x}$, are lying in that plane, and that the points are represented in CCS_1 and CCS_2, respectively, then the co-planar constraint is expressed by ${}^2\mathbf{x}^T\ \mathbf{E}\ {}^1\mathbf{x} = 0$ where $\mathbf{E}$ is the essential matrix. $\mathbf{E} = {}^2\mathbf{R}_1[{}^1\mathbf{t}_2]_\times$.
- This co-planarity leads also to a constraint of corresponding points in pixel coordinates: ${}^2\underline{\mathbf{p}}^T\ \mathbf{F}\ {}^1\underline{\mathbf{p}} = 0$. Here, $\mathbf{F}$ is the fundamental matrix. $\mathbf{F} = \mathbf{K}_2^{-T}\ \mathbf{E}\ \mathbf{K}_1^{-1}$.
- If we have a specific point ${}^1\underline{\mathbf{p}}$ in the first image, then the corresponding point in the second image must be located on the line defined by $\mathbf{F}\ {}^1\underline{\mathbf{p}} = 0$. This line is the epipolar line associated with ${}^1\underline{\mathbf{p}}$.

In our discussion, the camera 1 is the reference camera, and its role differs from the one of camera 2. Of course, we can reverse the role. By transposing (14) and (18) we get immediately the following results:

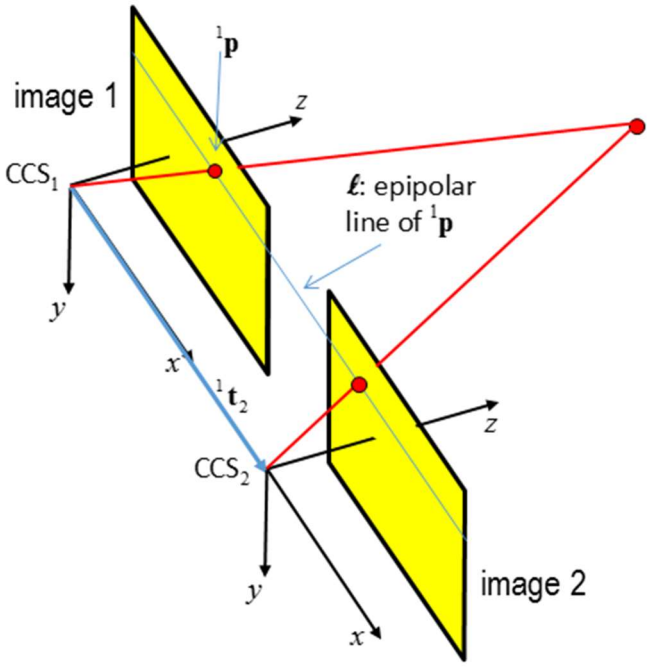
$$\begin{aligned} {}^1\mathbf{x}^T\ \mathbf{E}^T\ {}^2\mathbf{x} &= 0 \\ {}^1\underline{\mathbf{p}}^T\ \mathbf{F}^T\ {}^2\underline{\mathbf{p}} &= 0 \end{aligned} \qquad (19)$$

3 Stereo rectification

The set-up with the two identical cameras that are exactly aligned has the most straightforward and easiest epipolar geometry. Creating such a set-up facilitates the processing of the stereo images much. However, in practice the cameras are never perfectly aligned. This section introduces techniques to virtually rotate the cameras and virtually change the calibration matrices so as to bring back the epipolar geometry to its easiest state. This is called stereo rectification.

3.1 A fully aligned camera

For a fully aligned stereo set-up with a left and a right camera, the orientations of the cameras are equal ${}^1\mathbf{R}_2 = \mathbf{I}$ and ${}^2\mathbf{R}_1 = \mathbf{I}$, and the base line is oriented horizontally, i.e. ${}^1\mathbf{t}_2 = \begin{bmatrix} t_x & 0 & 0 \end{bmatrix}^T$. The horizontal directions of both image planes are parallel to the base line. The two optical axes are parallel. And they are orthogonal to the base line.



In this situation, the essential matrix is (see (10) and (12)):

$$\mathbf{E} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad (20)$$

We have used here the property of the essential matrix that a scaling factor β is not significant. That is $\beta\mathbf{E} \triangleq \mathbf{E}$. From (20) we see immediately that the null space of $\mathbf{E}$ is $\begin{bmatrix} \alpha & 0 & 0 \end{bmatrix}^T$, i.e. the x-axis. This is in accordance with the fact that the epipoles are lying on the x-axis at an infinite distance.

The fundamental matrix follows from (17). For a fully aligned stereo set-up, we require that the cameras are the same: $\mathbf{K}_1 = \mathbf{K}_2 = \mathbf{K}$. In that case, substitution of (20) in (17) yields:

$$\mathbf{F} = \mathbf{K}^{-\mathrm{T}} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{K}^{-1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{21}$$

where again it has been used that scaling doesn't matter: $\mathbf{F} \triangleq \beta\mathbf{F}$ for any $\beta \neq 0$.

The null space of this matrix, which equals the epipole in the first image, is ${}^{1}\underline{\mathbf{q}} = \begin{bmatrix} \alpha & 0 & 0 \end{bmatrix}^{\mathrm{T}}$. The transposed matrix $\mathbf{F}^{\mathrm{T}}$ has the same null space. Thus, the epipole of the second image is also ${}^{2}\underline{\mathbf{q}} = \begin{bmatrix} \alpha & 0 & 0 \end{bmatrix}^{T}$. These results are in accordance with the expectation that the epipoles are in the horizontal direction at an infinite distance.

3.2 Rectification

To transform two stereo images such that they become fully aligned, we need (a) to change the virtual calibration matrices so as to make them equal, and (b) to rotate the cameras such that they are oriented in the same direction.

There is one direction in the set-up that constraints the alignment orientation of the camera pair: the direction of the base line ${}^{1}\mathbf{t}_2$. We assume that the cameras are mounted left and right next to each other so that the x-component in ${}^{1}\mathbf{t}_2$ is dominant. We have to align the horizontal axes (rows of pixels) of the cameras with the base line. The other direction, the vertical axes (columns of pixels), can be chosen freely. This is because we can rotate the cameras along the base line and still maintain the requirements for a full alignment. Once the vertical direction has been selected, the third direction, the optical axis, is fixed as it should be orthogonal to the new image planes.

The first step of rectification is the construction of a rotation matrix ${}^{a}\mathbf{R}_1$ that, applied to the CCS_1, aligns the first image to the base line. To do so, we define:

$${}^{a}\mathbf{R}_1 = \begin{bmatrix} \mathbf{u}_x^{\mathrm{T}} \\ \mathbf{u}_y^{\mathrm{T}} \\ \mathbf{u}_z^{\mathrm{T}} \end{bmatrix} \tag{22}$$

The x-axis of the new coordinate system should be aligned to the vector ${}^{1}\mathbf{t}_2$. Thus ${}^{a}\mathbf{R}_1 {}^{1}\mathbf{t}_2 \propto \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^{T}$. Consequently:

$$\mathbf{u}_x = \frac{{}^{1}\mathbf{t}_2}{\left\| {}^{1}\mathbf{t}_2 \right\|} \qquad \mathbf{u}_y^{\mathrm{T}} {}^{1}\mathbf{t}_2 = 0 \qquad \mathbf{u}_z^{\mathrm{T}} {}^{1}\mathbf{t}_2 = 0 \tag{23}$$

Apart from the orthogonality constraint $\mathbf{u}_y \perp \mathbf{u}_x$, the direction of new y axis can be chosen freely. A reasonable choice would be to select it such that the direction of the optical axis is preserved as much as possible. This is obtained by:

$$\mathbf{u}_y = \mathbf{u}_x \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{{}^{1}t_x^2 + {}^{1}t_y^2}} \begin{bmatrix} -{}^{1}t_y \\ {}^{1}t_x \\ 0 \end{bmatrix} \tag{24}$$

Upon this choice, the third direction follows from:

$$\mathbf{u}_z = \mathbf{u}_x \times \mathbf{u}_y \tag{25}$$

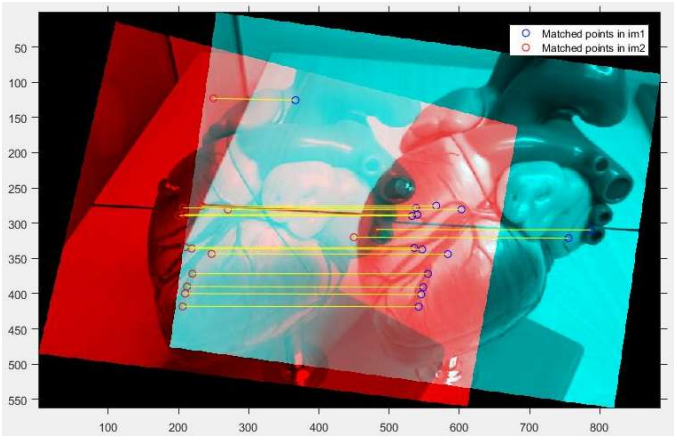
The alignment of the first camera is accomplished by the homography:

$${}^{a}\mathbf{H}_1 = \mathbf{K}_1 {}^{a}\mathbf{R}_1 \mathbf{K}_1^{-1} \tag{26}$$

To rectify the second image, we first undo the rotation of camera 2 relative to camera 1, i.e. ${}^{1}\mathbf{R}_2$. Next, we apply the alignment to the base line as was done with camera 1, i.e. ${}^{a}\mathbf{R}_1$, finally we correct for the difference in calibration matrix. In full, the homography becomes:

$${}^{a}\mathbf{H}_2 = \mathbf{K}_1 {}^{a}\mathbf{R}_1 {}^{1}\mathbf{R}_2 \mathbf{K}_2^{-1} \tag{27}$$

This expression not only rotates camera 2, but also sets its camera calibration matrix equal to camera 1.



An example of image rectification is shown above. It can be seen that all corresponding points are now on the same row. The images are truly rectified. In this new stereo set-up, the base line vector is given by:

$${}^{1}\mathbf{t}_2^{rect} = \begin{bmatrix} T & 0 & 0 \end{bmatrix}^{T} \quad \text{with} \quad T = \sqrt{t_x^2 + t_y^2 + t_z^2} \tag{28}$$

and the orientations are aligned: ${}^{1}\mathbf{R}_{2}^{rect} = \mathbf{I}$.

4 Stereo matching

Depth information is encoded in the image positions of two corresponding points. Finding these corresponding points is called stereo matching. This section addresses *dense stereo* in which the goal is to find the corresponding point for each pixel in the first image, and perhaps also vice versa. In its simplest form, stereo matching can be defined as:

> *Given a pixel* ${}^{1}\mathbf{p} = \begin{bmatrix} n & m \end{bmatrix}^{\mathrm{T}}$ *in the first image, determine the corresponding pixel* ${}^{2}\mathbf{p} = \begin{bmatrix} u & v \end{bmatrix}^{\mathrm{T}}$ *in the second image. Do this for all pixels in the first image.*

Complications arise from the fact that sometimes pixels might not have a corresponding pixel at all due to occlusion, and that pixels miss texture so that matching is hardly possible.

The task of stereo matching looks like a 2D search problem, but there are a few constraints to which any corresponding pair of points must comply. The most important one is the *epipolar constaint*. The corresponding point ${}^{2}\mathbf{p}$ must be located on the epipolar line associated with ${}^{1}\mathbf{p}$. This reduces the 2D search problem to a 1D search problem.

Once rectified, all epipolar lines are horizontal so that ${}^{1}\mathbf{p}$ and ${}^{2}\mathbf{p}$ are on the same row of the image, i.e. $m = v$. The correspondence is given by the *disparity* D. If (n,m) and $(u,m) = (u,v)$ are corresponding points, then the pixel shift:

$$D \overset{def}{=} n - u \tag{29}$$

is the disparity of (n,m). Assuming that each pixel (n,m) has a corresponding point, the disparity is defined for each pixel, and the so-called *disparity map* $D(n,m)$ is introduced. This map decodes for each pixel the depth.

Stereo matching is very often based on the so-called *CBA:* the *constant brightness assumption*, The CBA states that corresponding points have identical color or grey level. Suppose that $f(n,m)$ and $g(u,v)$ are the color images or intensity (grey level) images associated with camera 1 and camera 2, respectively. Then the CBA states, that:

$$f(n,m) = g(n - D(n,m), m) \tag{30}$$

From radiometry we know that the CBA is true for diffusely reflecting (Lambertian) surfaces. However, it might not be true in other cases such as glossy surfaces at locations showing highlights, e.g. the reflection of a light source. Hence, in a conditioned stereo set-up, we might want to avoid glossy spots by using diffuse illumination as much as possible.

4.1 Window-based stereo matching

Equation (30) alone to find for a given (n,m) the disparity $D(n,m)$ does not work out. Due to noise, and due to small deviations from the CBA, an exact solution will not exist. But if we build in a tolerance, i.e.:

$$\left| f(n,m) - g(n + D(n,m), m) \right| \leq \text{threshold} \tag{31}$$

then multiple solutions might exist that satisfies (31).

Window-based matching (synonym: block-based matching) adopts a second assumption: the colors in the vicinity of a point (n,m) in the image 1 are similar to the colors in the vicinity of the corresponding point $(n + D(n,m), m)$ in image 2. Usually, a *window* is defined as a rectangular area surrounding a pixel. Suppose that the window size is $(2K+1) \times (2L+1)$, then the window of (n,m) encompasses the set of pixels:

$$\left\{ (n+k, m+\ell) \;\middle|\; k = -K, \cdots, K \text{ and } \ell = -L, \cdots, L \right\}$$

Let us vectorize the colors or intensities of such a set of pixels by stacking them in one large column vector. Denote such a vector by $\mathbf{f}(n,m)$ and $\mathbf{g}(u,v)$, respectively. These vectors have dimension $3(2K+1)(2L+1)$ in case of RGB colors, and $(2K+1)(2L+1)$ in case of intensities.

For a given pixel (n,m) in the first image, window-based stereo matching boils down to regarding $\mathbf{f}(n,m)$ as a template which must be found back on the same row $v = m$ in the second image. For that purpose, a similarity criterion should be defined. Often used criteria are the NCC (normalized cross correlation), the SSD (sum of squared differences), and the SAD (sum of absolute differences):

$$\begin{aligned} NCC(\mathbf{f},\mathbf{g}) &= \frac{\mathbf{f}^T \mathbf{g}}{\|\mathbf{f}\|_2 \, \|\mathbf{g}\|_2} \\ SSD(\mathbf{f},\mathbf{g}) &= \|\mathbf{f} - \mathbf{g}\|_2^2 = \sum_i (f_i - g_i)^2 \\ SAD(\mathbf{f},\mathbf{g}) &= \|\mathbf{f} - \mathbf{g}\|_1 = \sum_i |f_i - g_i| \end{aligned} \tag{32}$$

$\| \; \|_p$ denotes the L_p-norm defined as follows:

$$\|\mathbf{y}\|_p \overset{def}{=} \left(\sum_n |y_n|^p \right)^{\frac{1}{p}} \qquad (33)$$

The pseudo code for window-based matching in the case of SSD becomes:

Algorithm : "window-based stereo matching using SSD"

input :

images: $f(n,m)$ and $g(n,m)$ with size $N \times M$

window size: K and L

maximum disparity: $D_{\max}$

for all row $m = 1, \cdots, M$

for all columns $n = 1, \cdots, N$

1. for all allowable disparities $d = 1, \cdots, D_{\max}$

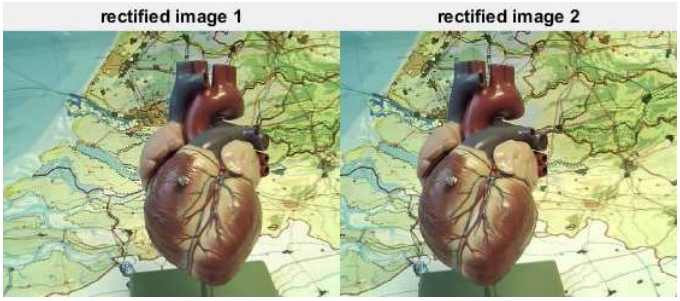
   Calculate SSD: $ssd(d) = \|\mathbf{f}(n,m) - \mathbf{g}(n-d,m)\|^2$

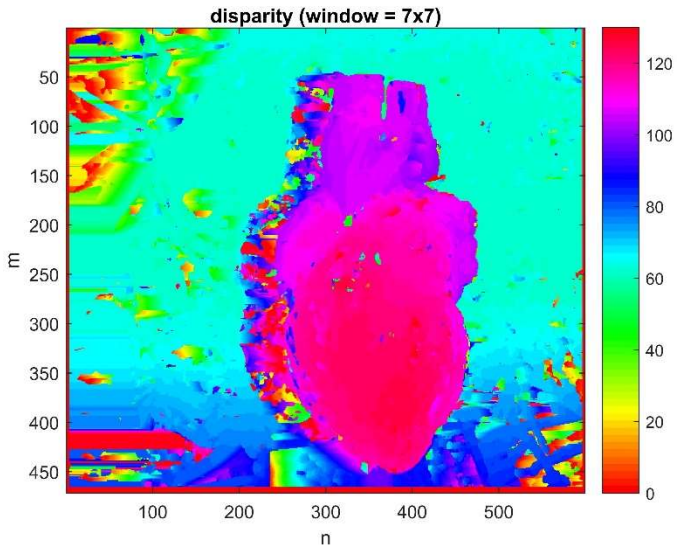
2. $D(n,m) = \arg\min_{d=1,\cdots,D_{\max}} \{ssd(d)\}$

Output :

the disparity map $D(n,m)$

As an example, the figure below shows two rectified stereo images. Application of window-based stereo matching using the SSD criterion yields a disparity map as shown.





The performance of window-based matching is often poor. We already mentioned that the CBA does not hold everywhere. More seriously, the second assumption is also often violated. This is for a number of reasons:

- Occlusion:
  A surface patch that might be visible in one image, but might not be visible in the other image. In the example above, this happens on the left side of the front object.
- Discontinuities of the depth:
  The surface patch might be near the occluding boundary of an object, i.e. near the edge. If the window partly overlaps the edge, then that part of the window does not even approximately equals that part of the other window.
- Perspective distortion:
  The windows of two corresponding points are only identical if the surface is fronto-parallel to the image planes. That is, if all pixels in the window share the same depth. This is rarely the case, and if it is not, the perspective projection of the 3D surface patch surrounding the 3D point will be different for the two images.
- Low texture
  In order to have it work fine, sufficient texture must be available within the window. A short coming of that will lead to errors. In the example this occurs on the left top and the right bottom.

4.2 Constraints on the solution

We consider a rectified pair of stereo images. Camera 1 is the reference camera as before. Assuming that camera 1 is situated left from camera 2, the disparity map has some constraints to which it must comply. These constraints help to find a solution.

1. Disparity range constraint
   The disparity of a 3D point is inversely proportional to its depth. The range of allowable disparities is therefore limited and can be determined from the minimum and maximum depth in the scene. See Section 5. If the depth is at infinity, the minimum disparity is zero. Thus, negative disparities are not allowed.
2. Symmetry constraint
   If a pixel in image 1 correspond to a pixel in image 2, then the vice verse is also true.
3. Uniqueness constraint
   A pixel in image 1 can correspond at most to one pixel in image 2. An exception of this constraint exists, but this occurs rarely.
4. Disparity smoothness constraint

The disparities of two pixels that are in each others vicinity do not differ much. This is not a hard constraint as it is not true at the occluding boundary of an object.

5. Ordering constraint

   At smooth surfaces the ordering of points is preserved along the epipolar lines. That is, if $({}^1\mathbf{p}_a, {}^2\mathbf{p}_A)$, $({}^1\mathbf{p}_b, {}^2\mathbf{p}_B)$, and $({}^1\mathbf{p}_c, {}^2\mathbf{p}_C)$ are 3 pairs of corresponding points on the same row (epipolar line), and if the order in image 1 is *a,b,c.*, then the order of the points in image 2 is *A,B,C.*

A pixel (n,m) in image 1 has a corresponding pixel in image 2 given by $(n-D(n,m),m)$, See (29). If n is incremented, i.e. we consider the point $(n+1,m)$ in image 1, then the x-coordinate of the corresponding point in image 2 becomes $n+1-D(n+1,m)$. The uniqueness constraint and the ordering constraint boil down to:

$$n+1-D(n+1,m) > n-D(n,m) \tag{34}$$

If in (34) the left-hand-side would be less than the right-hand-side, the ordering constraint would be violated. If they would be equal, the uniqueness constraint is violated. From (34):

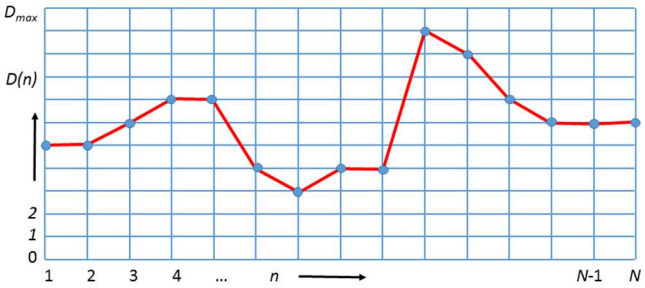
$$D(n+1,m) < 1+D(n,m) \tag{35}$$

This inequality will be used in the next section.

4.3 Dynamic programming

Dynamic programming is a branch of optimization methodology in which an optimal solution of a complex problem is sought by combining the solutions of smaller sub-problems. One of the problems that can be solved by dynamic programming is to find the optimal path in a trellis. See figure below. The problem is defined as follows. A path always starts at $n=1$ and ends at $n=N$. Such a path is defined by the sequence $D(n)$. By assumption, $D(n)$ is restricted to a finite number of integers in the range from 0 to $D_{\max}$. The figure below shows an example of a path in the trellis.

Dynamic programming assumes that costs $G[D,n]$ are assigned to each grid point of the trellis. In addition, costs $T[D(n),D(n-1)]$ are also assigned to transitions. These costs depend only on $D(n)$ and its predecessor $D(n-1)$, but they do not depend explicitly on n. The full cost of a path equals:



$$J = \sum_{n=1}^{N} G[D(n),n] + \gamma T[D(n),D(n-1)] \tag{36}$$

Because $D(0)$ does not exist, the first term $T[D(1),D(0)]$ is defined as zero. The design parameter γ is a fixed weight that determines the balance between the two types of costs. The goal is to find the path with minimum J.

This statement of the optimization problem fits perfectly well on the estimation of disparities if the image is processed row by row. For a given row m we estimate the sequence $D(n,m)$ with $n=1,\cdots,N$. We use the shorthand notation $D(n)$ for $D(n,m)$. For the cost $G[\,]$ we use an appropriate similarity criterion such as mentioned in Section 4.1. For instance, the SSD:

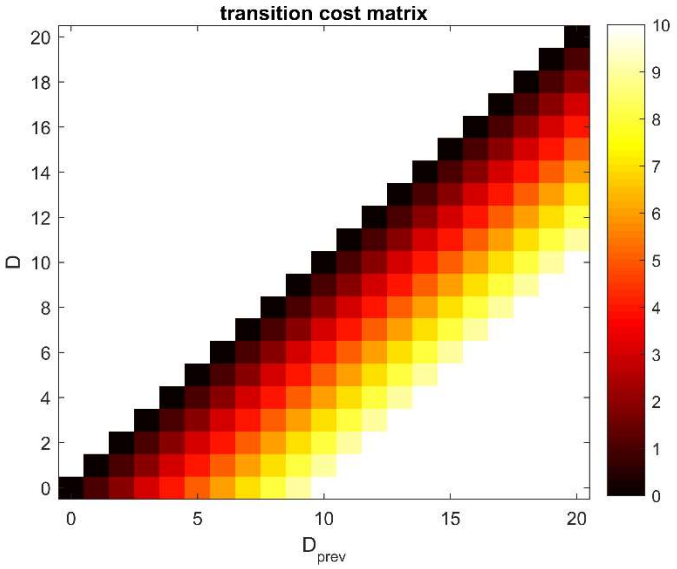
$$G[D,\mathrm{n}] = ssd(D) = \|\mathbf{f}(n,m) - \mathbf{g}(n-D,m)\|^2 \tag{37}$$

For the transition costs, we use some heuristics based on the constraints mentioned in Section 4.2. The cost function $T[\,]$ could be defined as follows:

$$T[D,D_{prev}] = \begin{cases} |D-D_{prev}| & \text{if } -a \le D \le 1+D_{prev} \\ a & \text{elsewhere} \end{cases} \tag{38}$$

See figure below. This cost function (cost matrix, actually) favors small decrements of D. An example is given below for $D_{\max}=20$ and $a=10$.

Finding the optimal path using a brute force method is undoable: the number of different paths is: $(1+D_{\max})^N$. Fortunately, the search can be implemented much faster. For that purpose, introduce the so-called *optimal return function* $V(n,D)$. Suppose that $\{\hat{D}_n(i) \text{ with } i=n,\cdots,N\}$ are the optimum paths starting at given $(n,\hat{D}_n(n))$ and ending at $i=N$. Note that there are $D_{\max}+1$ of these paths since $\hat{D}_n(n)$ can vary between 0 and $D_{\max}$. The optimal return function, evaluated at a grid point (n,D), is the cost of such an optimal path starting from (n,D). The cost is expressed as:



$$V(n,D)=\sum_{i=n}^{N} G\left[i,\hat{D}_n(i)\right]+\gamma\sum_{i=n+1}^{N} T\left[\hat{D}_n(i),\hat{D}_n(i-1)\right] \quad (39)$$

with $\hat{D}_n(n)=D$

The essence of dynamic programming is that if we know $V(n,D)$ for all D, then it is easy to calculate $V(n-1,D)$. If we start with $V(N,D)$:

$$V(N,D)=G[N,D] \quad \text{for} \quad D=0,\cdots,D_{\max} \quad (40)$$

then we can recursively find the other optimal return functions:

$$V(n,D)=\min_{d=0,\cdots,D_{\max}}\left\{G[n,D]+T[d,D]+V(n+1,d)\right\} \quad (41)$$

Once the optimal return function has been calculated by applying (41) backward, i.e. for $n=N-1,N-2,\cdots,1$, the optimal path is found by tracing the optimal return function forwardly. The algorithm of dynamic programming as a solution for stereo matching is given below.

The array $U(n,D)$ that is filled in step 3.2 stores the optimal transitions, so that tracing the optimal path in step 5 becomes easy. An example of a result of dynamic programming is shown below. It can be seen that most of the rows show smooth disparity except at the right boundary of the front object. The occluded areas show more or less random disparities as could be expected.

Algorithm : "stereo matching based on dynamic programming"

input :

images: $f(n,m)$ and $g(n,m)$

window size: K and L

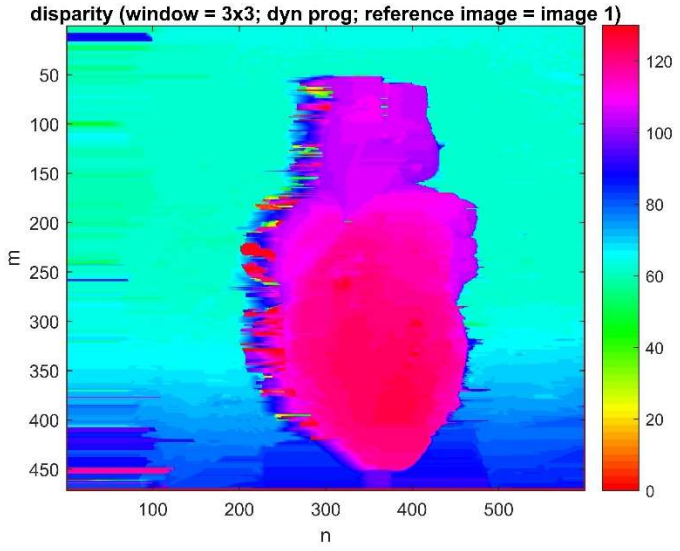
maximum disparity: $D_{\max}$

transition cost matrix: $\mathbf{T}$ with size $(1+D_{\max})\times(1+D_{\max})$

for all row $m=1,\cdots,M$

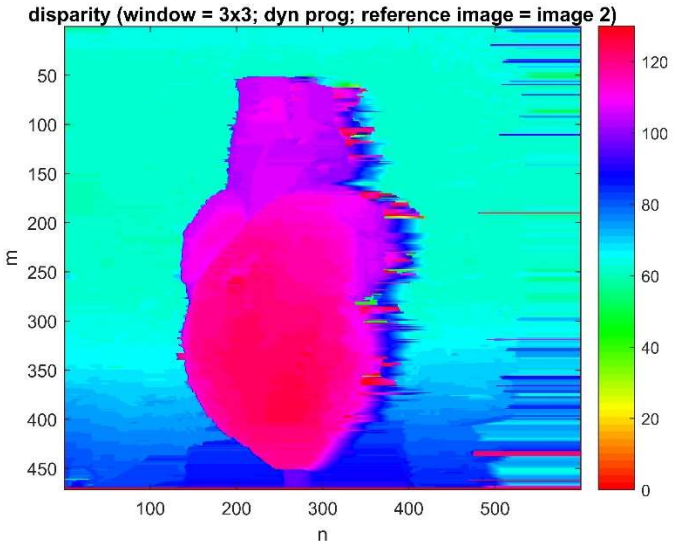
1. for all columns $n=1,\cdots,N$, and all disparities $d=0,\cdots,D_{\max}$:
   $G(n,d)=ssd(d)=\left\|\mathbf{f}(n,m)-\mathbf{g}(n-d,m)\right\|^2$
2. for all disparities $d=0,\cdots,D_{\max}$:
   Initialize the optimal return function $V(N,d)=G(N,d)$
3. for all columns $n=N-1,\cdots,1,$ and all disparities $d=0,\cdots,D_{\max}$:
   3.1 $V(n,D)=\min_{d=0,\cdots,D_{\max}}\left\{G(n,D)+T(d,D)+V(n+1,d)\right\}$
   3.2 $U(n,D)=\arg\min_{d=0,\cdots,D_{\max}}\left\{G(n,D)+T(d,D)+V(n+1,d)\right\}$
4. Select the path found in step 3 with minimal cost:
   $D_{opt}(1)=\arg\min_{d=0,\cdots,D_{\max}}\left\{V(1,d)\right\}$
5. Trace the optimal path: for all column $n=2,\cdots,N$
   $D_{opt}(n)=U\left(n,D_{opt}(n-1)\right)$
6. $D(n,m)=D_{opt}(n)$

Output : disparity map $D(n,m)$



Symmetry constraint

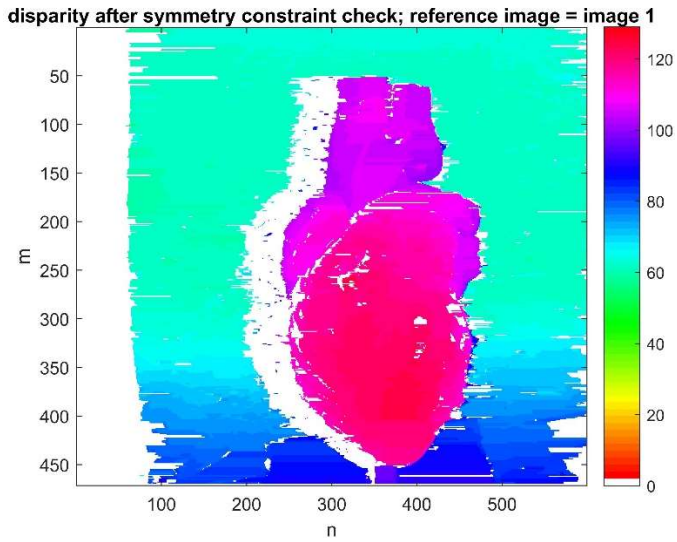
In the example, shown above, the disparity was calculated with image 1 as the reference image. The disparity of image 2 with reference to image 1 has been calculated. Of course, the roles can also be reversed. The result is shown below.



As can be seen, the disparities are quite similar except that now the occlusion problem shows up on the other side. This, however, offers the possibility to perform a consistency check on all disparities based on the symmetry constraint. If the disparities with reference to image 1 are denoted by $D_1(n,m)$ and those with reference to image 2 by $D_2(u,m)$ then according to (29):

$$D_1(n,m) = D_2(n - D_1(n,m), m) \tag{42}$$

We can test this condition with a tolerance, in Matlab called *DistanceThreshold*, to label disparities as being unreliable. The result is shown below. Here the tolerance was 3 pixels. Unreliable pixels are labeled white.



4.4 Global methods

The limitation of dynamic programming, as established in the previous section, is that it processes the image row by row ignoring the fact that depth, and thus disparity, not only has context along the rows, but also in all other directions. A global cost criterion takes 2D neighborhoods into account. Suppose that $\mathrm{N}(n,m)$ is the neighborhood of pixel (n,m), i.e. a 4-neighborhood or a 8-neighborhood, then a cost is assigned to disparity differences between $D(n,m)$ and all disparities in the neighborhood. The cost of the disparity of a single pixel is:

$$G[n,m] + \gamma \sum_{(i,j)\in \mathrm{N}(n,m)} T_{i,j}\left[D(n,m), D(i,j)\right] \tag{43}$$

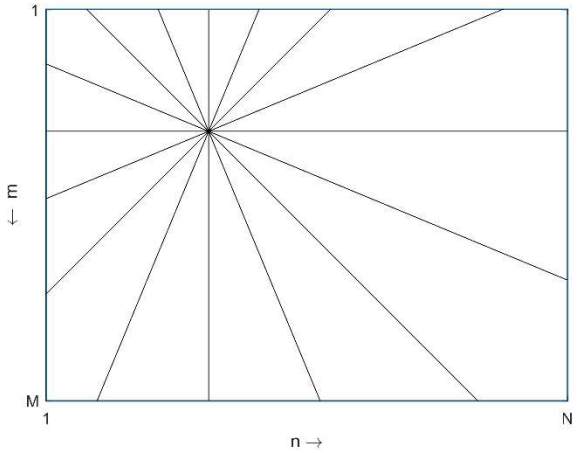
so that the full costs become:

$$J = \sum_{n=1}^{N}\sum_{m=1}^{M}\left(G[n,m] + \gamma \sum_{(i,j)\in \mathrm{N}(n,m)} T_{i,j}\left[D(n,m), D(i,j)\right]\right) \tag{44}$$

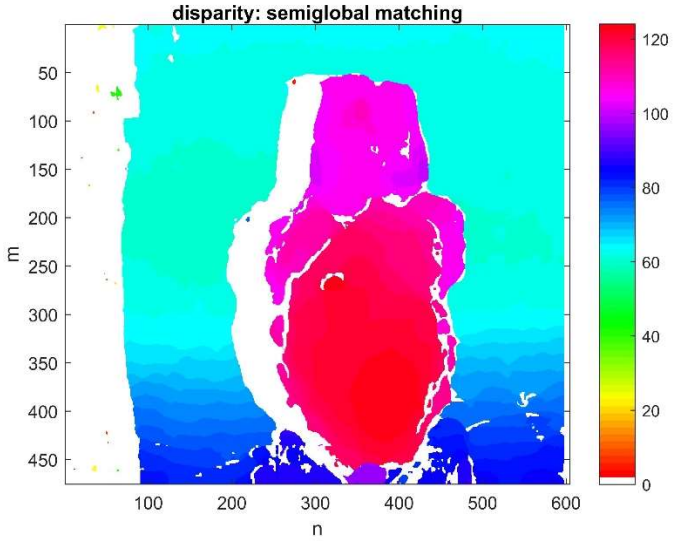
The first term is image-driven and takes care of a solution that is consistent with the observed image data. The second term takes care of a solution in which the disparity is consistent, e.g. smooth within its whole context.

Optimization of (44) is not easy. Different techniques exist such as graph-cut algorithms and simulated annealing, but these techniques are often computationally expensive.

A semi-global optimization is proposed by Hirschmüller. In addition to the row-oriented dynamic programming approach, other orientations are also considered.



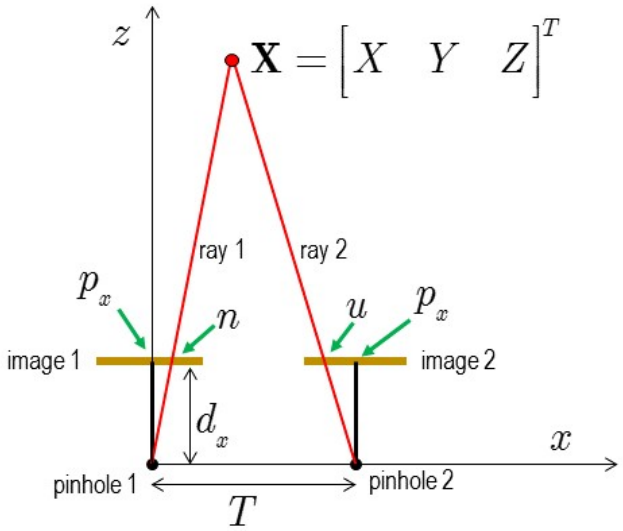
The figure above shows 8 scan lines which all share a common point. Dynamic programming is applied to each scan line resulting in 8 optimal return functions $V_\ell(n,m,D)$ with $\ell = 1,\cdots,8$, of which the minimum is selected. The method is called *SemiGlobal Matching* (SGM) [1]. The result of SGM applied to the example images is shown below.



Compared to the row-by-row approach of simple dynamic programming, the disparity map is much more consistent in all directions, although at the bottom of the front object the SGM performs worse.

5 From disparity to 3D point clouds

The disparity map decodes the depth of the pixels in image 1. To encode the depth consider the figure below showing a XZ-view of the aligned stereo set-up. Knowing the disparity D_1 at a given pixel position (n,m) the goal is to find the 3D position $\mathbf{X}=\begin{bmatrix}X & Y & Z\end{bmatrix}^{\mathrm{T}}$ of the 3D point. This point is to be expressed in CCS_1 coordinates.



In this set-up, the equation of ray 1 is given by:

$$Z=\frac{d_x}{n-p_x}X \tag{45}$$

p_x is the x-component of the principal point (image center). We note that the fraction $(n-p_x)/d_x$ is dimensionless as all variables are expressed in pixel distances.

Ray 2, expressed in CCS_1 has the equation:

$$Z=\frac{d_x}{u-p_x}(X-T) \tag{46}$$

Substitution of (45) in (46) and solving for Z yields

$$Z=\frac{d_xT}{n-u}=\frac{d_xT}{D_1} \tag{47}$$

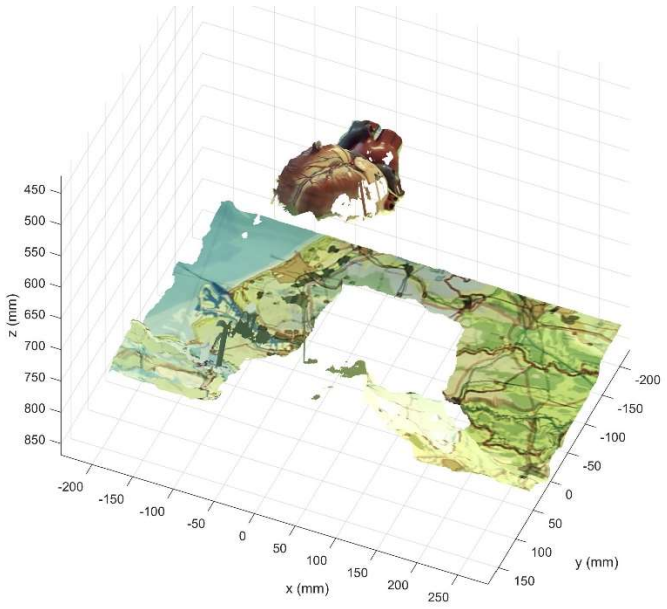
X and Y are found by the perspective projection equations. This results in:

$$X=\frac{n-p_x}{d_x}Z \quad \text{and} \quad Y=\frac{m-p_y}{d_y}Z \tag{48}$$

Application of (47) and (48) to the whole disparity map $D_1(n,m)$, yields a *point cloud*:

$$\mathbf{X}_{cloud}(n,m)=\begin{bmatrix}X(n,m) & Y(n,m) & Z(n,m)\end{bmatrix}^{\mathrm{T}} \tag{49}$$

In the disparity maps, shown above, white pixels are considered unreliable. Here, these pixels are identified by the assignment of a disparity of -1. The corresponding points in the point cloud are also unreliable. These points should be labelled as such by, for instance, by assigning a NaN (not a number) to $Z(n,m)$.



The figure above is a visualization of a *3D surface mesh* (Section 6.5) derived from the point cloud coming from the disparity map shown in Section 4.4.

6 Functions in Matlab

6.1 Stereo camera calibration

Calibration of a stereo camera set-up is most conveniently done with the app *Stereo Camera Calibrator*. The resulting object is of type `stereoParameters`. It contains the following properties:

- `cameraParameters1` and `cameraParameters2`:
  These are objects of type `cameraParameters`. They contain the intrinsic parameters of camera 1 and camera 2, respectively. These include the camera calibration matrix and the lens deformation parameters. The camera calibration matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ are the transposed version of the parameter `IntrinsicMatrix`.
- `RotationofCamera2`:
  This represents the 3x3 rotation matrix ${}^2\mathbf{R}_1$, i.e. the orientation of camera 1 expressed[2] in the coordinate system of camera 2. Note, however, that Matlab consistently uses the transposed form, so that this matrix actually equals ${}^2\mathbf{R}_1^T$ (which is ${}^1\mathbf{R}_2$).
- `TranslationofCamera2`:
  This represents the 1x3 translation vector ${}^2\mathbf{t}_1$, i.e. the position of the origin of camera 1 expressed in the coordinate system of camera 2. It is given in transposed form, so that this vector actually equals ${}^2\mathbf{t}_1^T$.

The app also provides the uncertainty of the calibration parameters. These are given in an object of class .`stereoCalibrationErrors`.

Instead of using the app, the calibration can also be done with functions. For that the following suit is available:

- `imageSet`
- `detectCheckerboardPoints`
- `generateCheckerboardPoints`
- `estimateCameraParameters`
- `displayErrors`

See the example provided in `EstimateAndDisplay-StereoCalibrationStandardErrorsExample.m`.

6.2 Rectification

The function `rectifyStereoImages` takes as an input two stereo images, and the stereo calibration parameters. The output are rectified images that are first compensated for nonlinear lens deformations. The output format can be 'full' meaning that the output images contains the full input data. Or, it can be 'valid' meaning that the output images are cropped to the largest common rectangle containing valid pixels.

As discussed in Section 3.2, rectification is not an unambiguous process. Cameras can be virtually rotated along the base line and still preserve the required properties for a full alignment. Unfortunately, the documentation of the function `rectifyStereoImages` does not clarify which of the many valid solutions for rectification has been implemented. Also, many valid choices for the camera calibration matrix of the output image can be made. It remains undocumented which calibration matrix has been selected.

6.3 Stereo matching

Stereo matching is provided in the function `disparity`. The input consists of two rectified images. The output is a disparity map. By default, disparity uses semiglobal matching using a 15x15 window, the SAD measure, and a disparity range of [0 64]. Optionally, only window-based (block-matching) is used.

Different strategies can be applied to detect unreliable disparities:

- If the contrast, estimated by the Sobel edge operator, is too large the pixel is probably on an edge, i.e. on the boundary of an object.
- If the SAD has not a clear minimum within the allowable range of disparities, the found disparity is unreliable.
- If the symmetry constraint (left-right and right-left disparities should be the same) is violated outside a tolerance zone, the disparity is assumed to be unreliable.
- If the texture within the window is too small, the disparity is unreliable.

[2] This is unlike the text in the Matlab documentation which suggests that the orientation and translation of camera 2 is given with respect to camera 1.

Unreliable disparities are indicated in the disparity map by setting their value to `-realmax`, i.e. the largest negative number.

6.4 Point clouds

The function `reconstructScene` takes as input a disparity map and the stereo calibration parameters. It produces a point cloud which is an MxNx3 array containing for each disparity the 3D coordinates of the reconstructed points If a disparity is unreliable, the coordinates of the corresponding 3D point gets a `NaN`. A point can also get `Inf` or `-Inf`. This could happen if the disparity is zero.

Matlab also defines a special class for point clouds called `pointCloud`. This class includes many methods and supporting functions for manipulation of the point cloud, for instance:

- `pcshow`: plot the point cloud.
- `pcdenoise`: remove noise.
- `pcdownsample`: downsample the point cloud.
- `pcregrigid`: register two point clouds using ICP.
- `pctransform`: rigid geometrical transform.
- `pcmerge`: merge two point clouds.
- `pcnormals`: estimate normal vectors of point cloud.
- `pcfitplane`: fit a plane to the point cloud.
- `pcfitsphere`: fit a cylinder to the point cloud.
- `pcfitcylinder`: fit a cylinder to the point cloud.

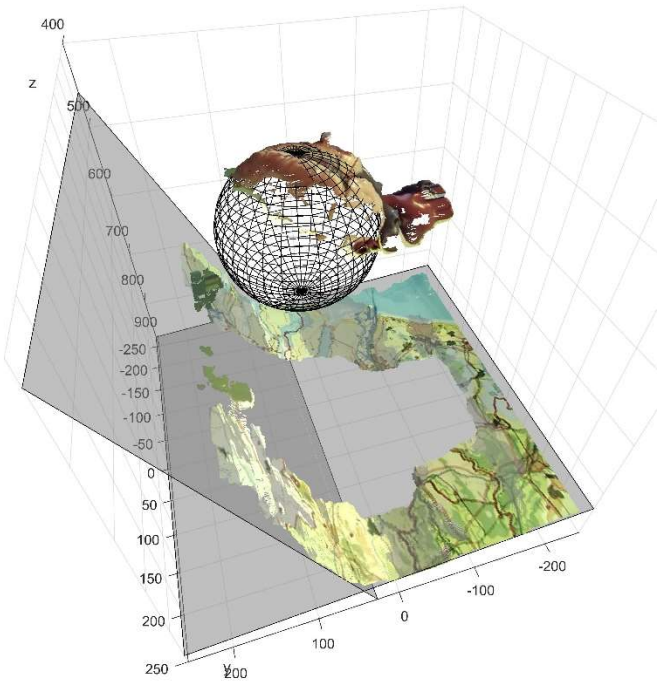
An application of plane fitting and sphere fitting is shown below. The scene consisted of approximately two planes which are correctly found. The model of the heart, which is not really a sphere, has been reasonably approximated, even though only a small part of the surface is available.

6.5 Creating 3D surface meshes

A 3D surface mesh is formed by a set of adjacent triangles. A triangle, called a *face*, is defined by the 3D positions of its 3 corners. Corners are called *vertices*. The 3D surface mesh is represented by:

- A list of vertices.
- A connectivity list.

Suppose that there are P vertices, then the vertex list is a Px3 array. The id of a specific vertex is given by its row subscript of the array. Suppose that there are Q triangles, then the connectivity list is a Qx3 array. Each row in the connectivity list defines a triangle by giving the ids of the 3 vertices.

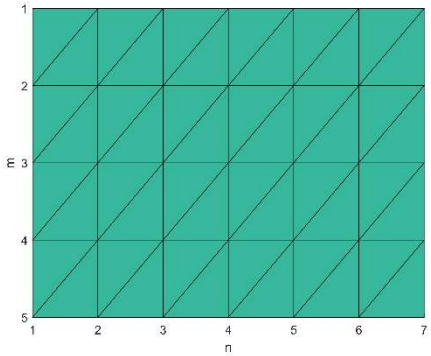


In Matlab, the class `triangulation` exists to represent such a surface mesh. In this class, the properties `Points` and `ConnectivityList` hold the two lists. Several methods and supporting functions exist to manipulate the mesh. For instance, functions for plotting the mesh are `trimesh` and `trisurf`.

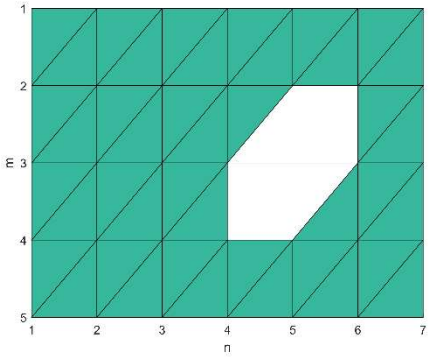
In order to transform a point cloud into a mesh, we assume that we have:

- `disparityMap`: the NxM disparity map.
- `pc`: the NxMX3 point cloud.
- `unreliable`: An NxM map indicating for each pixel that the disparity is unreliable.
- `J1`: The NxMx3 rectified image.

First we have to define a connectivity structure. This can be done in the orthogonal grid of the image by splitting each rectangular pixel area into two triangles. For an example of a 7x5 image, see below.

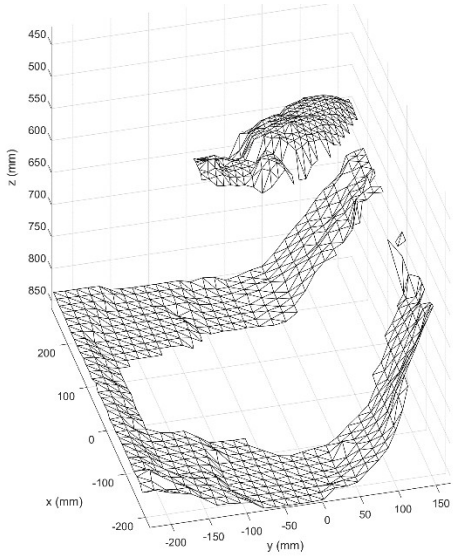


Next, we have to remove all the triangles which contain an unreliable point. Suppose that pixel (5,3) is unreliable, then removing all triangles connected to that point yields a structure with a hole:



Once this structure has been made, we can associate each point (n,m) to a 3D point `pc(m,n,:)` from the point cloud with associated color `J1(m,n,:)`.

Matlab example code of this procedure is provided below. This code results in the surf plot shown in Section 5. A mesh plot is produced by changing a few lines of code:




```
%% Matlab code for creating a 3D surface mesh

%% create a connectivity structure
[M, N] = size(disparityMap);              % get image size
res = 2;                                  % resolution of mesh
[nI,mI] = meshgrid(1:res:N,1:res:M);      % create a 2D meshgrid of pixels, thus defining a resolution grid
TRI = delaunay(nI(:),mI(:));              % create a triangle connectivity list
indI = sub2ind([M,N],mI(:),nI(:));        % cast grid points to linear indices

%% linearize the arrays and adapt to chosen resolution
pcl = reshape(pc,N*M,3);                  % reshape to (N*M)x3
J1l = reshape(J1,N*M,3);                  % reshape to (N*M)x3
pcl = pcl(indI,:);                        % select 3D points that are on resolution grid
J1l = J1l(indI,:);                        % select pixels that are on the resolution grid

%% remove the unreliable points and the associated triangles
ind_unreliable = find(unreliable(indI));% get the linear indices of unreliable 3D points
imem = ismember(TRI(:),ind_unreliable); % find indices of references to unreliable points
[ir,~] = ind2sub(size(TRI),find(imem)); % get the indices of rows with refs to unreliable points.
TRI(ir,:) = [];                           % dispose them
iused = unique(TRI(:));                   % find the ind's of vertices that are in use
used = zeros(length(pcl),1);              % pre-allocate
used(iused) = 1;                          % create a map of used vertices
map2used = cumsum(used);                  % conversion table from indices of old vertices to the new one
pcl = pcl(iused,:);                       % remove the unused vertices
J1l = J1l(iused,:);
TRI = map2used(TRI);                      % update the ind's of vertices

%% create the 3D mesh
TR = triangulation(TRI,double(pcl));      % create the object

%% visualize
figure;
TM = trimesh(TR);                         % plot the mesh
set(TM,'FaceVertexCData',J1l);            % set colors to input image
set(TM,'Facecolor','interp');
% set(TM,'FaceColor','red');              % if you want a colored surface
set(TM,'EdgeColor','none');               % suppress the edges
xlabel('x (mm)')
ylabel('y (mm)')
zlabel('z (mm)')
axis([-250 250 -250 250 400 900])
set(gca,'xdir','reverse')
set(gca,'zdir','reverse')
daspect([1,1,1])
axis tight
```

References

The theory and techniques, explained in this chapter, can be found back in many textbooks on Computer Vision [2-6]

[1] H. Hirschmuller, “Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information,” in Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02, 2005, pp. 807-814.

[2] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 3rd ed.: Thomson-Engineering, 2008.

[3] R. Hartley, and A. Zisserman, *Multiple View Geometry in Computer Vision*: Cambridge University Press, 2003.

[4] D. A. Forsyth, and J. Ponce, *Computer Vision: A Modern Approach*, 2nd ed.: Pearson Education Limited, 2012.

[5] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision: From Images to Geometric Models*: SpringerVerlag, 2003.

[6] R. Szeliski, *Computer Vision: Algorithms and Applications*: Springer-Verlag New York, Inc., 2010.