VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

STEREO BASED 3D FACE RECONSTRUCTION

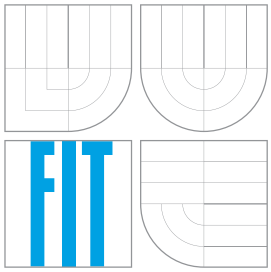
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MILAN FALEŠNÍK

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D REKONSTRUKCE MODELU LIDSKÉ TVÁŘE S POMOCÍ DVOU KALIBROVANÝCH WEBKAMER

STEREO BASED 3D FACE RECONSTRUCTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MILAN FALEŠNÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MAREK ŠOLONY
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá vytvořením programu pro rekonstrukc trojrozměrného modelu lidské tváře za pomocí dvojice kamer. Pro řešení využívá výpočtu hloubkové mapy a následný převod do trojrozměrného prostoru. Využívá Viola-Jones detektor pro detekci tváře. Využívá knihovny OpenCV a také částečně PCL.

Abstract

This thesis presents a system for reconstruction of three-dimensional model of human face by using pair of cameras. To solve this problem, depth map calculation is used and then the depth map is transformed into three-dimensional space. It uses Viola-Jones detector for detecting the face. Uses libraries OpenCV and partially PCL.

Klíčová slova

stereovize, model lidské tváře, výpočet hloubkové mapy, stereo rekonstrukce, OpenCV, PCL, PvAPI

Keywords

stereo vision, human face model, disparity calculation, stereo reconstruction, OpenCV, PCL, PvAPI

Citace

Milan Falešník: Stereo Based 3D Face Reconstruction, bakalářská práce, Brno, FIT VUT v Brně, 2013

Stereo Based 3D Face Reconstruction

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Marka Šolonyho. Další informace mi poskytli Dr. Andrea Lagorio, prof. Enrico Grosso a prof. Massimo Tistarelli

.....
Milan Falešník
July 31, 2013

Poděkování

Chtěl bych poděkovat vedoucímu práce a pracovníkům laboratoře počítačového vidění v Porto Conte Ricerche pod Università degli studi di Sassari (Prof. Massimo Tistarelli, Dr. Andrea Lagorio, Prof. Enrico Grosso) za cenné rady při tvorbě této práce. Speciálně Dr. Lagoriovi za rady ohledně rekonstrukce a Prof. Grossovi za jeho materiály o modelu kamery a kalibrace

Contents

Chapter 1

Introduction

We live in the era of computers which develops very fast. In just about 70 years, computers were able to shrink from big, house-sized monsters with their own power plants into a small device which fits inside USB flash case. As the time had gone, computers have learned more and more skills. From calculating ballistic trajectories to recognition of spoken word. Many industrial areas are dominated by computers, as they do not make mistakes and do not want to raise their salaries. In many of these areas, computers have to see the outer world to check whether everything is correct. This is the area of *computer vision*, subdiscipline of artificial intelligence.

Computer vision tries to mimic human ability to see - see depth, see shapes and recognize them, see the movement etc. As the costs of IT equipment and cameras go down, their popularity grows up and new possibilities of usage appear. Suitably equipped computer then can replace, for example, a sculptor. It's possible to recognize and save shape of human face.

Computer model of human face (or any part of human body in general) can be useful for example for medical use. Nowadays specialized machines for human face model reconstruction exist, although their price is comparable to the price of new car (figure 1.1). With right software equipment, doctors are able to plan e.g. facial surgeries etc.

When the 3D printers will be available for a reasonable price, it will be possible to create real model of human face by ordinary people.

I cannot forget about computer games players. Some games offer modification of player's avatar body (including face). It is possible to personalize the character's face by setting the properties of the face (e.g. Second Life). If you wanted a *realistic* model of your own face in majority of other games, you met with unsuccess unless you had some experience with 3D modeling, texturing and of course, game data hacking.

So, when the regular equipment is cheap nowadays, what if computer games' manufacturers offered placement of own face model into their games? Facial model is not an usual thing to use, but I am sure that more uses will appear for it.

This work will show you the possibility how to create a human face model beginning with a simple cheap stereo camera setup design. Then we will move to the analysis of some methods that can be used for recognition of the shape of particular object (human face in this case), which will be succeeded with implementation and testing of designed system.

Figure 1.1: Visualization of model from state-of-art face modelling machine.

Chapter 2

A stereo camera setup

This chapter will present the pinhole camera model and calibration of the camera. Then the stereo system will be presented together with the camera placement possibilities and explanation of the possibility of reconstruction of the model of the face. The calibration model presented here is actually a bit different from the one used in the program (OpenCV uses somewhat more complex one calculating also with additional distortions), but the principle is the same.

2.1 Lens system model and pinhole camera model

Based on [9] and [8]. For thin lenses:

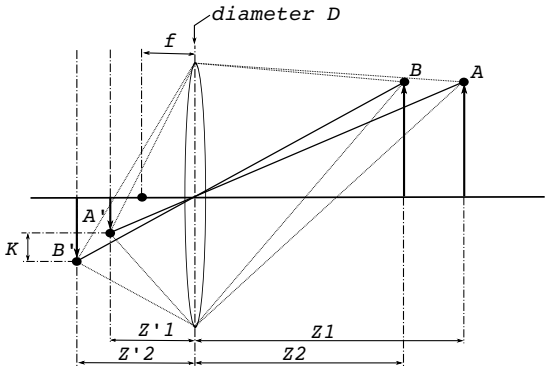


Figure 2.1: Lens model.

$$\frac{1}{Z'_1} + \frac{1}{Z_1} = \frac{1}{f}$$

$$\frac{1}{Z'_2} + \frac{1}{Z_2} = \frac{1}{f}$$

$$|Z'_1 - Z'_2| = |\frac{f}{Z_1 - f} \cdot \frac{f}{Z_2 - f} \cdot (Z_1 - Z_2)|$$

If the object A is clearly in focus, object B is out of focus.

$$K = \frac{D}{Z'_2}|Z'_1 - Z'_2| = D \cdot |\frac{f}{Z_1 - f}| \cdot |\frac{Z_1 - Z_2}{Z_2}|$$

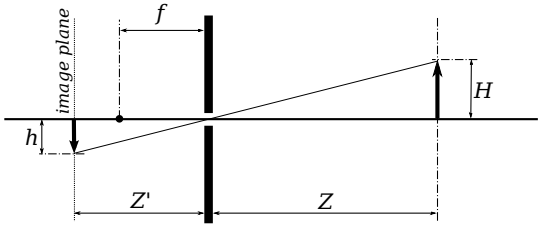


Figure 2.2: Pinhole model.

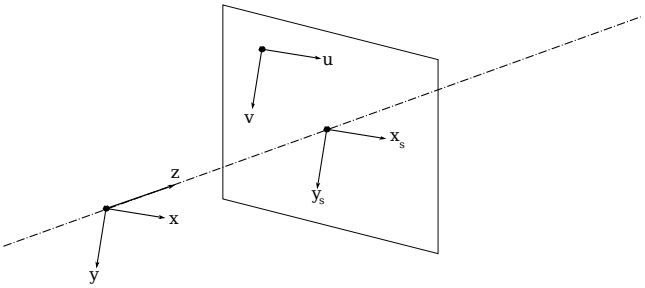


Figure 2.3: Reference system. Axis z is perpendicular to image plane.

If $D \to 0$, then pinhole camera model is obtained. For this system, everything is in focus.

The position of the image plane depends on z. If $z \gg f$, then $z' \cong f$. This implies:

$$|h| = |\frac{f \cdot H}{z}|$$

For the pinhole model, a triad of reference is often used.

- (u, v) - image coordinates
- (x_s, y_s) - translated system in the image
- (x, y, z) - reference triad of the camera

There are following relationships:

$$x_s = \frac{u - u_0}{K_u}$$

$$y_s = \frac{v - v_0}{K_v}$$

(u_0, v_0) represent the intersection between optical axes in the image plane. (K_u, K_v) represent the independent scaling on two image axes.

$$x = x_s \frac{z}{f}$$

$$y = y_s \frac{z}{f}$$

f is the focal distance. When these relationships are combined together:

$$x = z \frac{u - u_0}{f \cdot K_u}$$

$$y = z\frac{v - v_0}{f \cdot K_v}$$

There are 11 parameters to calibrate:

- Intrinsic (5)
  - f
  - u_0, v_0
  - K_u, K_v (pixel dimensions)
- Extrinsic (6)
  - Position reference triad
  - Orientation reference triad

2.2 Calibration of single camera [7]

The problem consists in general from:

- Finding a relationship between image coordinates (u, v) and world coordinates (x_m, y_m, z_m). Matrix M is called *perspective matrix*.
- Finding intrinsic and extrinsic parameters of the camera together (matrices A and C).

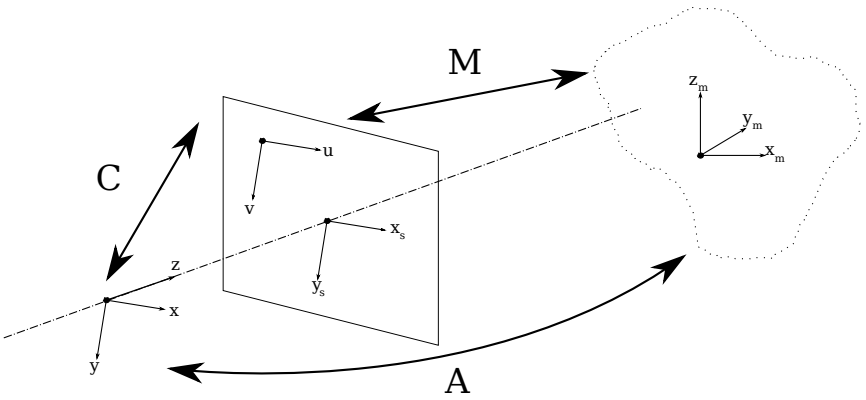


Figure 2.4: Relationships between coordinate systems.

Homogenous coordinates:

$$\begin{vmatrix} S_u \\ S_v \\ S \end{vmatrix} = \begin{vmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{vmatrix} \cdot \begin{vmatrix} x_c \\ y_c \\ z_c \\ 1 \end{vmatrix} = C \cdot \begin{vmatrix} x_c \\ y_c \\ z_c \\ 1 \end{vmatrix}$$

This equals to pin-hole model.

$$\begin{vmatrix} x_c \\ y_c \\ z_c \\ 1 \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} x_m \\ y_m \\ z_m \\ 1 \end{vmatrix} = \begin{vmatrix} \underline{a}_1 & a_{14} \\ \underline{a}_2 & a_{24} \\ \underline{a}_3 & a_{34} \\ 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} x_m \\ y_m \\ z_m \\ 1 \end{vmatrix} = A \begin{vmatrix} x_m \\ y_m \\ z_m \\ 1 \end{vmatrix}$$

These relationships do not provide the solution sought by itself (x_c, y_c, z_c are not known). By combining these two relationships the following relationship is obtained:

$$\begin{vmatrix} S_u \\ S_v \\ S \end{vmatrix} = \begin{vmatrix} \alpha_u \underline{a}_1 + u_0 a_3 & \alpha_u a_{14} + u_0 a_{34} \\ \alpha_v \underline{a}_2 + v_0 a_3 & \alpha_v a_{24} + v_0 a_{34} \\ \underline{a}_3 & a_{34} \end{vmatrix} \cdot \begin{vmatrix} x_m \\ y_m \\ z_m \\ 1 \end{vmatrix} = \begin{vmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{vmatrix} \cdot \begin{vmatrix} x_m \\ y_m \\ z_m \\ 1 \end{vmatrix} =$$

$$= \begin{vmatrix} \underline{m}_1 & m_{14} \\ \underline{m}_2 & m_{24} \\ \underline{m}_3 & m_{34} \end{vmatrix} \cdot \begin{vmatrix} x_m \\ y_m \\ z_m \\ 1 \end{vmatrix} = M \cdot \begin{vmatrix} x_m \\ y_m \\ z_m \\ 1 \end{vmatrix}$$

Note that the number of parameters to estimate in M are **12**. Because access into the image is always (u, v), 2 equations per point are obtained.

$$u = \frac{m_{11} x_m + m_{12} y_m + m_{13} z_m + m_{14}}{m_{31} x_m + m_{32} y_m + m_{33} z_m + m_{34}}$$

$$v = \frac{m_{21} x_m + m_{22} y_m + m_{23} z_m + m_{24}}{m_{31} x_m + m_{32} y_m + m_{33} z_m + m_{34}}$$

If all parameters are scaled uniformly, then there are **11** real parameters. $5\frac{1}{2}$ points are needed.

In the real world, many points are used and estimation using least squares is calculated. With a bit of algebra it's possible to place them for example like this:

$$Q \cdot \underline{b} = \underline{d}$$

- Q - (x_m, y_m, z_m) of different points
- $\underline{b}$ - unknowns m_{11}, m_{12}, ...
- $\underline{d}$ - (u, v) of different points

$$\underline{b} = (Q^T Q)^{-1} Q^T \cdot \underline{d}$$

This is the solution of the least squares estimation.

Note that the solution for the perspective matrix does not supply intrinsic/extrinsic parameters. However the values found for m_{ij} compensate for:

- Scaling
- Rotation and translation of the origin of the camera
- Unortogonality of the axes of the sensor (*skew*)
- Different scaling of the axes of the sensor (*shear*)

For finding the intrinsic/extrinsic parameters it's possible to assume that the rotational part of matrix A is *ortonormal*. The steps are:

- Finding the correct scaling factor

$$M = \omega \cdot \begin{vmatrix} \alpha_u \underline{a}_1 + u_0 \underline{a}_3 & \alpha_u a_{14} + u_0 a_{34} \\ \alpha_v \underline{a}_2 + v_0 \underline{a}_3 & \alpha_v a_{14} + v_0 a_{34} \\ \underline{a}_3 & \underline{a}_{34} \end{vmatrix}$$

Because $\omega \underline{a}_3 = \underline{m}_3$ impose a condition of unitary norm:

$$\omega = \sqrt[2]{\underline{m}_3 \cdot \underline{m}_3^T}$$

Elimination of the scaling factor of M is obtained by division by ω:

$$M' = \frac{1}{\omega} M$$

- Searching for other parameters after calculating M'. Because of the vectors' $\underline{a}_i$ ortogonality:

$$\underline{a}_3 = \underline{m}'_3$$
$$u_0 = \underline{m}'_1 \cdot \underline{m}'_3$$
$$v_0 = \underline{m}'_2 \cdot \underline{m}'_3$$
$$\alpha_u = |\underline{m}'_1 \times \underline{m}'_3|$$
$$\alpha_v = |\underline{m}'_2 \times \underline{m}'_3|$$
$$\underline{a}_1 = \frac{1}{\alpha_u} \cdot (\underline{m}'_1 - u_0 \underline{m}'_3)$$
$$\underline{a}_2 = \frac{1}{\alpha_v} \cdot (\underline{m}'_2 - u_0 \underline{m}'_3)$$
$$a_{14} = \frac{1}{\alpha_u} \cdot (m'_{14} - u_0 m'_{34})$$
$$a_{24} = \frac{1}{\alpha_v} \cdot (m'_{24} - v_0 m'_{34})$$
$$a_{34} = m'_{34}$$

2.3 Model and calibration of stereoscopic system [9]

Note that there are **6** levels of liberty. Solution with R matrix (rotational/translational matrix with 12 parameters):

$$\begin{vmatrix} x_l \\ y_l \\ z_l \\ 1 \end{vmatrix} = R \cdot \begin{vmatrix} x_r \\ y_r \\ z_r \\ 1 \end{vmatrix}$$

From the pinhole model:

$$\frac{x_l}{z_l} = \frac{u_l - u_{0l}}{\alpha_{u_l}}; \frac{x_r}{z_r} = \frac{u_r - u_{0r}}{\alpha_{u_r}}$$
$$\frac{y_l}{z_l} = \frac{v_l - v_{0l}}{\alpha_{v_l}}; \frac{y_r}{z_r} = \frac{v_r - v_{0r}}{\alpha_{v_r}}$$

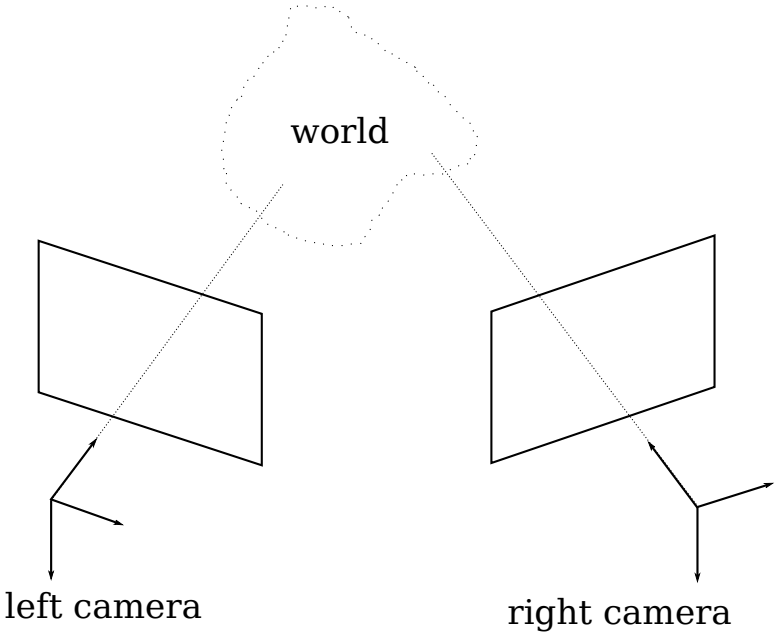


Figure 2.5: System representation.

By substituting x_l, y_l, x_r, y_r, more general case is obtained:

$$\frac{u_l - u_{0l}}{\lambda_{ul}} \cdot z_l = r_{11}\frac{u_r - u_{0r}}{\lambda_{ur}} z_r + r_{12}\frac{v_r - v_{0r}}{\lambda_{vr}} z_r + r_{13} z_r + r_{14}$$

$$\frac{v_l - v_{0l}}{\lambda_{vl}} \cdot z_l = r_{21}\frac{u_r - u_{0r}}{\lambda_{ur}} z_r + r_{22}\frac{v_r - v_{0r}}{\lambda_{vr}} z_r + r_{23} z_r + r_{24}$$

$$z_l = r_{31}\frac{u_r - u_{0r}}{\lambda_{ur}} z_r + r_{32}\frac{v_r - v_{0r}}{\lambda_{vr}} z_r + r_{33} z_r + r_{34}$$

It's possible to demonstrate that by eliminating the unknowns z_l and z_r, one linear equation is obtained. For simplicity, let's say that:

$$\lambda_{ul} = \lambda_{ur} = \lambda_{vl} = \lambda_{vr} = \lambda$$

$$u_{0l} = u_{0r} = v_{0l} = v_{0r} = 0$$

By applying this, these equations are obtained:

$$u_l \cdot \frac{z_l}{z_r} = r_{11} u_r + r_{12} v_r + r_{13}\lambda + \frac{r_{14}}{z_r}\lambda$$

$$v_l \cdot \frac{z_l}{z_r} = r_{21} u_r + r_{22} v_r + r_{23}\lambda + \frac{r_{24}}{z_r}\lambda$$

$$\lambda \cdot \frac{z_l}{z_r} = r_{31} u_r + r_{32} v_r + r_{33}\lambda + \frac{r_{34}}{z_r}\lambda$$

Parameters z_l, r_{14}, r_{24}, r_{34} are linked by a relationship of proportionality to z_r. This system is not linear in 14 unknowns $r_{11}..r_{34}$, z_l and z_r. By eliminating z_l and z_r, one linear equation with 12 unknowns $r_{11}.., r_{34}$ is obtained for every point in image.

2.4 Stereo vision

You may have noticed that when you had covered one of your eyes, you had slight problem with approximating the distance of the objects you see. That is because you've lost your stereo vision ability (stereopsis) with only one eye active. When you use only one eye, it behaves just like a regular camera - it creates a 2-D representation of 3-D world around you inside the eye and your brain can just approximate the real-world coordinates from informations it gathered throughout your life.

Uncover both eyes and now you can safely see the distances of the objects. You still see 2-D pictures physically, but your brain combines those two images into representation that you receive by your sight with knowledge of your eyes's parameters.

The key to the stereo vision is that when you see the same scene with N (where $N > 1$) receivers (cameras or eyes), you can compute the *depth* of the scene - the shape of the objects. As you see at Figure 2.6, just two receivers are enough to recognize depth of the

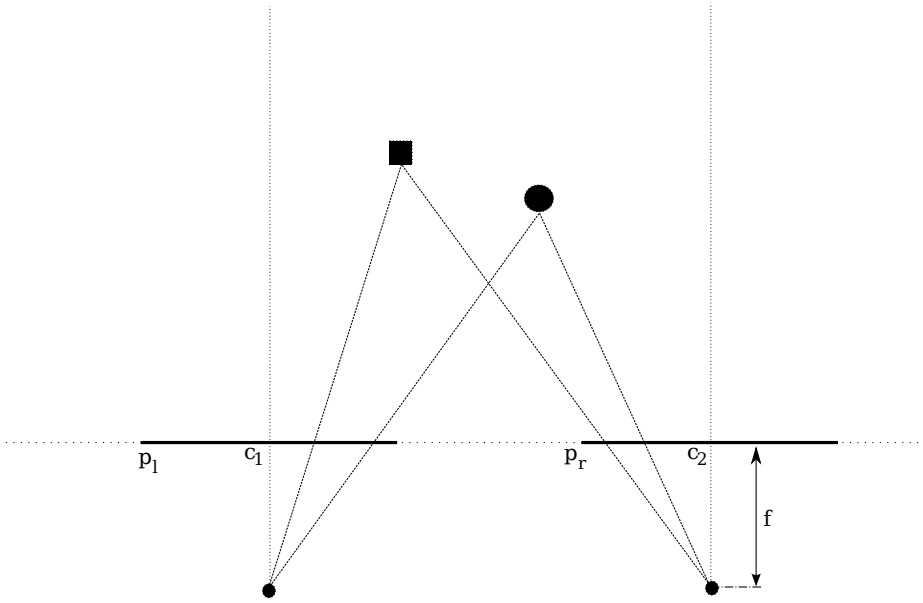


Figure 2.6: Basic principle of triangulation.

scene. For precise reconstruction of somewhat more complex scene (for example a skull), more receivers would have to be used in a different setup. However, this is not needed for my purposes, as human face's surface can be considered continuous, therefore reconstructable with just 2 cameras.

The essential question for computer stereo vision is „Which *token* from image B corresponds to a given token from image A?" Because there are several possibilities for selecting the token in image B in general, we say the correspondence problem is *ambiguous* [8] (chapter 6). This raises a number of other questions. Which tokens, which features or which constraints can be used to reduce this ambiguity? The most important constraint is *epipolar constraint*. Applying this constraint is called *rectification*. OpenCV also adds undistortion in this phase, because cheap cameras suffer from radial and tangential distortion. After rectification, all possible matches for given token in the first image are on the same line in the second image. Basically, all methods for gathering disparity map mentioned in this work use this constraint.

2.5 Stereo cameras

There are several possible basic configurations of stereo camera [2].

- *Converged configuration* - Cameras are placed apart by the inter-axial distance. Then the axes are converged by small angle. This angle is sometimes called „toe-in“.
- *Parallel configuration* - Similar like *converged* but with zero „toe-in“ so the cameras's axes are kept parallel.
- Cameras can also be positioned freely, but this makes the calibration more difficult.

In general - the bigger the distance between cameras is, the bigger disparity difference is emitted with the depth change. With the parallel camera setup it also makes less space for recognizing depth because this space is made by overlapping the viewing spaces of both cameras. With converged setup, this problem is somewhat smaller because of the abovementioned „toe-in“. The camera placement is not as important as for example for stereoscopic images. There won't be any stereoscopic image for end user produced where cameras have to be placed specifically to avoid headaches and other side effects of watching stereoscopic imagery. This will be subject of experiments later (6.2.2).

Illumination of the scene also has to be considered. An example - if the face is illuminated from one side, it's possible to see a shadow casted by the nose on the other side, which can affect the reconstruction algorithm as it makes surface lose its original texture. For these purposes, enough light is needed to cancel the shadows - but not too much because too light surfaces can lose their texture detail also and again confuse the algorithm.

When dealing with common web cameras, it has to be fought with the fact that it's unlikely to make the cameras hardware synchronized, so the captured images may be shifted by some small amount of time. This is not a big problem as it's possible to make the captured object to be still. But the capture parameters may make problems as the cameras have built-in automatic exposure and white balance control, which is sometimes not possible to override, if for example one camera was used as the reference one and the second camera would take settings from the first. Even with the distance the same as it is of the human eyes, cameras can have different white balance setting or exposure based on scene's illumination. This can confuse the algorithm used for estimating the disparity.

I will use two kinds of camera for the experiments. First pair of cameras are my own pair of webcams, which represent the „lower-end“ stereo equipment. Second pair are Prosilica GE680 cameras, representing the higher end. Although these capture just black-and-white images in VGA, they have very good sensor and interchangeable lens with manual control over aperture and focus. This allows making precise camera setup with parameters that do not automatically change.

Chapter 3

Face recognition and extraction

After capturing the imagery it's needed to find and cut out the face to know which part of the disparity map will be reconstructed into the face model. I am using simpler approach using Viola-Jones detector [13] with cascades trained for human face, nose and eyes and I am using face approximation by an ellipse. I found better method which tracks the face edge and finds the true face border in the image [12], but there was a lack of informations about implementation.

3.1 Face recognition

As I mentioned before, I am using Viola-Jones detector. By using Viola-Jones detector with face cascades I obtain rectangular area where the face is located. But because this cannot say much about face geometry, another step is needed. So, inside of this rectangle, I apply another cascades for detection, nose and eyes. Center of the nose will be the center of the facial ellipse. By the distance between nose center and eyes center I estimate the vertical dimensions of the face. Horizontal axe of the ellipse is specified by detected face rectangle width. Vertical axe of the ellipse is specified by detected face rectangle height. But to prevent glitches caused by hair or chin, there are two so-called „cut-points“, which specify the minimum and maximum position on vertical axe. As a reference unit, distance between nose center and eyes center is taken:

$$m = NoseCenter_y - EyesCenter_y$$

Then the cut-points calculation is very easy:

$$Top_y = EyesCenter_y - m \cdot 1.4$$

$$Bottom_y = NoseCenter_y + m \cdot 2$$

The multiplying constants are selected heuristically, as these values proved to be enough good. However this method works best only with face facing directly towards the camera.

3.2 Face extraction

For the purposes of extracting the face, I use approach in „Monte Carlo“ style. There is a function, which checks whether given point is is within bounds of the cut-points and then it checks whether the given point lies inside of an ellipse given in centre of the face

rectangle and touching all four sides of the rectangle. During the reconstruction, algorithm iterates over all points in the rectangle, but actually uses only those, which are marked by the function as inside the face region.

Chapter 4

Disparity map

Disparity map represents the difference of position in x axis for each pixel, therefore specifying the depth layer where given point lies. Every point from original image is assigned a value representing its position in Z-axis. Real positions of all points in the disparity map can be obtained by reprojection using M perspective matrix (5.1). It's then possible to put all points into point cloud or generate triangles to obtain the model. Because of this, the quality of disparity map is essential for re-constructing the model of the scene. The best results will be obtained with stable, carefully set cameras and precise stereo calibration of them.

Figure 4.1: Good disparity map. This is how it should look. Smooth surface and no holes.

Figure 4.2: This disparity map is not usable, because area with nose and around eyes wasn't reconstructed well. However if you see this, you are on a good way, this probably signifies problem with illumination, but the reconstruction algorithm works well.

Figure 4.3: This disparity map is totally unacceptable.

This chapter will present some methods for gathering disparity maps from stereo setup and also methods to improve their quality. One can use algorithms, which are faster, but have lower quality of resulting disparity map, or slower but more accurate algorithms.

4.1 Block matching

The block matching algorithm calculates the matching costs for the correspondence of every pixel in left image f_L to all pixels in the searching region S in the right image f_R. The searching region is specified by the maximum magnitude of the disparity vectors. The matching cost $mc(i, j; m, n)$ of assigning pixel (i, j) of the left image to pixel (m, n) of the right image is equal to:

$$mc(i, j; m, n) = \sum_{k,l} |f_L(i + k, j + l) - f_R(m + k, n + l)|$$

where $-N \leqslant k, l \leqslant N$ and (m, n) belongs to corresponding searching region of (i, j). The matching cost is calculated over blocks of size $(2 \cdot N + 1) * (2 \cdot N + 1)$, where N is user selectable. [10]

This method is implemented in OpenCV as *cv::StereoBM*

4.2 Semi-global block matching

This algorithm aims to minimize the following global energy function E for a disparity image D.

$$E(D) = \sum_p (C(p, D_p) + \sum_{q \in N_p} P_1 I[|D_p - D_q| = 1] + \sum_{q \in N_p} P_2 I[|D_p - D_q| > 1])$$

The minimized function produces a perfect disparity map with smoothing governed by parameters P_1 and P_2, however, minimizing the function for a 2D image space is an NP-complete problem. The semi-global matching function approximates the 2D minimization by performing multiple 1D, or linear, minimizations. The matching function aggregates costs on multiple paths which converge on the pixel under examination. Cost is computed for the disparity range specified by the minimum disparity and number of disparities parameters. By default, the matching algorithm aggregates costs for 5 directions. You can set the full dynamic programming parameter to true to force the algorithm to aggregate costs for 8 directions. [3]

This method is implemented in OpenCV as *cv::StereoSGBM*

4.3 Minimum-cut/Maximum-flow Stereo (MCS)

Also called *graph-cut* method [6]. Instead of processing the epipolar lines one by one, it takes all of them into account. This makes MCS method working in 2-D, thus having better quality of resulting disparity map. The basic principle is to extend the 4-neighbourhood of the each point in the space to the 6-neighbourhood, which makes it connected to adjacent epipolar lines. However, this makes dynamic programming unusable, so this method uses a graph to represent the viewing space. The bounding points, which are on the side facing the point of view, are all connected to the single point in the graph, the *source*. The bounding points on the opposite side of viewing space are connected to another single point in the graph, the *sink*.The next step is solving the *Maximum-flow* problem. There is a cost function, which is not trivial to obtain, but works for Lambertian surfaces[1]:

$$cost(\mathbf{p}'_0, d) = \frac{1}{N} \sum (\mathbf{v}(\mathbf{p}'_0, d) - \bar{\mathbf{v}}(\mathbf{p}'_0, d))^2$$

The set of edges that are saturated by the *Maximum-flow* represent a *Minimum-cut* of the graph. Resulting disparity surface is obtained with *Minimum-cut* on the solved graph. Because this method has very good results but it is time consuming, it is a subject of optimizations, for example in [5]

This method is implemented in OpenCV, but marked obsolete. However it can be still used.

4.4 Pre-processing of the disparity map

After capturing stereo image pair, image must be converted to grayscale. To prevent distortion caused by possible noise, median filter with element size 3 is applied. Contrast and brightness should be as much equal as possible in both of them, because disparity

[1] *Lambertian surface* is completely matte surface. Reflected light is same in all directions. Human face can be considered as a Lambertian surface for my purposes to keep things simple.

algorithms look for the same token in both images, as mentioned before (2.4). This is mainly important for cameras where it's not possible to control exposure and white balance (typically web-cams). During experiments I discovered, that equalizing the histogram of both images makes quite balanced results for this purpose.

4.5 Post-processing of the disparity map

Resulting disparity map's quality varies depending on used algorithm and scene. I identified 2 major flaws that occur in disparity maps:

- *Holes* - Sometimes the reconstruction algorithm fails to find correct disparity for certain region in image, which results in „black spots" with unspecified depth. This is usually caused by larger area of uniform colour. When filtered out, this makes holes ($z \to \infty$) in the resulting model. For solving this problem, morphological filtering can be used. By applying *closing* morphological filter (described in [11]) on resulting disparity map, lighter (correct) pixels spread into the „black spots" and fill them. This works perfectly for small holes. If there are bigger ones, the problem is usually caused with configuration of the reconstruction method or illumination. Block matching methods (4.1, 4.2) are prone to this flaw.
- *Jags* - Generated disparity map is not always smooth and generated model is subsequently very jaggy. Gaussian blur (11x11) showed to be solution in this case. When applied, almost all jags are gone and the face looks as it should look.

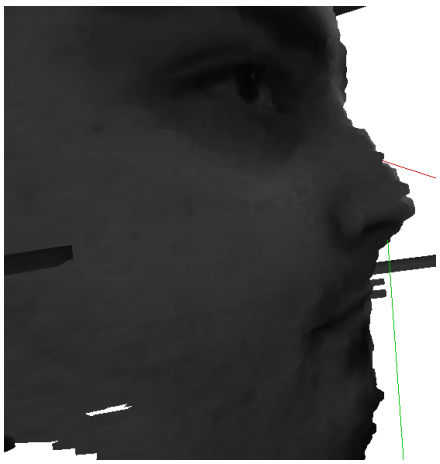


Figure 4.4: Without Gaussian blurring.

Figure 4.5: With Gaussian blurring.

After applying morphological filter and Gaussian blur, disparity map is ready for projection into a 3-D space.

Chapter 5

Model generation

After obtaining a desirable disparity map it's possible to proceed to the model generation. The tasks are:

- Reprojecting disparity map into 3-D space
- Using resulting points to reconstruct the scene

5.1 Reprojection

Disparity map (M_D here) contains integer values which say in which disparity layer specified point from source image lies. Let's show it on a simple $3 \cdot 3$ example:

$$M_D = \begin{vmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{vmatrix}$$

These informations aren't worth much for the scene reconstruction, so a reprojection takes place. Perspective matrix (M) is used for this purpose.

$$M = \begin{vmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{c_x - c'_x}{T_x} \end{vmatrix}$$

c'_x is the principal point x coordinate in right image. By using the M matrix, the M_D matrix is reprojected into the M_{3D} matrix. For each point in the source matrix M_D, this equation is applied [4]:

$$M \begin{vmatrix} x \\ y \\ d \\ 1 \end{vmatrix} = \begin{vmatrix} X \\ Y \\ Z \\ W \end{vmatrix}$$

3D coordinates of the point are obtained: $(X/W, Y/W, Z/W)$. After that, following matrix is obtained:

$$M_{3D} = \begin{vmatrix} (p_{11_X}, p_{11_Y}, p_{11_Z}) & (p_{12_X}, p_{12_Y}, p_{12_Z}) & (p_{13_X}, p_{13_Y}, p_{13_z}) \\ (p_{21_X}, p_{21_Y}, p_{21_Z}) & (p_{22_X}, p_{22_Y}, p_{22_Z}) & (p_{23_X}, p_{23_Y}, p_{23_z}) \\ (p_{31_X}, p_{31_Y}, p_{31_Z}) & (p_{32_X}, p_{32_Y}, p_{32_Z}) & (p_{33_X}, p_{33_Y}, p_{33_z}) \end{vmatrix}$$

This shows us an exact position in the world for each pixel (mapping $(u, v) \rightarrow (x, y, z)$) and these informations can be used for reconstruction.

5.2 Manual reconstruction

It's possible to use a very simple model reconstruction algorithm. Each 4 neighboring points can be taken to form 2 triangles. Let's look at $3 \cdot 3$ example:

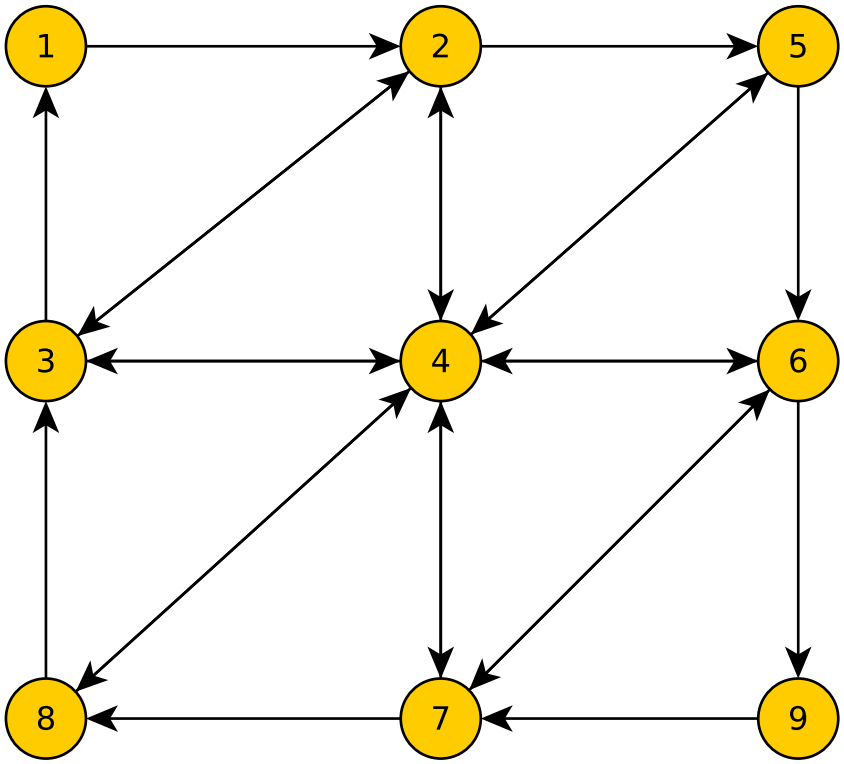


Figure 5.1: Points to triangles transformation.

From these points, triangles $(1, 2, 3)$, $(2, 4, 3)$, $(2, 5, 4)$, $(5, 6, 4)$, $(3, 4, 8)$, $(4, 7, 8)$, $(4, 6, 7)$ and $(6, 9, 7)$ are generated. Colour is taken from the original image for every vertex[1]. This method is very easy and by using it, it's quite straightforward to render the model in 3-D (e.g. by using OpenGL) or save into an easy format to use (e.g. STL[2]). It produces surprisingly good results for viewing. Number of triangles produced for a rectangle region is determined by this equation:

$$(w - 1) \cdot (h - 1) \cdot 2$$

where w is number of pixels in horizontal axis and h is number of pixels in vertical axis of the rectangle. The real number of generated triangles will be of course lower, as the face extracted is not taking all the space of the face rectangle (see more at chapter 3).

[1] *Vertex* is a point in the space. The same is the pixel in 2-D, the vertex is in 3-D

[2] *STL* - STereoLitography. Format widely used in 3-D printing with very simple structure

5.3 Reconstruction using PCL

By using PCL (6.1.3), things can be made more easy, but it's not always 100% win situation. Because as the name implies, the PCL works with point clouds, so instead of iterating over all neighboring quaternions of points, all points are put into the point cloud and then the normal estimation and reconstruction algorithms are used. Advantage is that there is less code, no need for programming the 3-D viewer and easier exporting to general formats, however the reconstructed surface quality is not as smooth as from the previous method. This is caused by breaking the neighbour binds and general loss of surface topology when the points are inserted into point cloud.

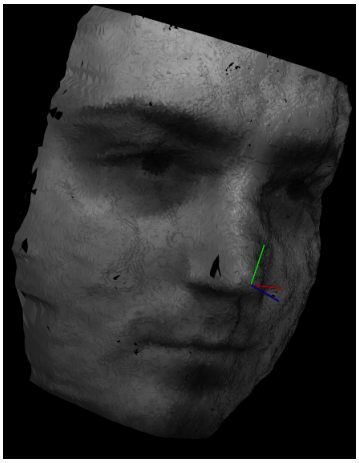


Figure 5.2: Render using PCL.

Chapter 6

Implementation and testing

Application was developed and tested on an Acer TimelineX 4820TG laptop (Core i5 430M 2.26 GHz, 8 GB RAM) in Fedora 18 operating system (64-bit). Because this laptop is equipped with a graphics card from AMD-ATI, no GPU acceleration from OpenCV's side was possible.

6.1 Frameworks and APIs used for developing

6.1.1 OpenCV

OpenCV is a computer vision framework, which covers the process of gathering image data and processing into a very nice API black box. It has a lot of implemented state-of-art techniques which the user of the framework can use for image processing. My interest takes turn towards its stereo vision and face recognition abilities. With OpenCV, I don't have to care about math behind the camera calibration, stereo calibration or undistorting images for getting rectified images. It also has some methods for gathering the disparity map implemented and I can use Haar cascade detector for detecting the face in the image with pre-trained data.

6.1.2 Qt

To be more user-friendly than command-line interface, I built the application with Qt4 toolkit in Qt Creator, because it's the only GUI toolkit I can use so far.

6.1.3 PCL

Point Cloud Library (see [1]) is a library designed for work with point clouds and it's also targeted at reconstruction. This library is very popular to use with Microsoft Kinect hardware. It can also be used in this case as described in 5.3.

6.1.4 PvApi

PvApi is a C API to communicate with cameras over 1Gbit Ethernet (*GigE protocol*). I extended this API a bit to be more usable in C++ environment and made it mutually usable with my webcams by using OOP. I used this protocol during my visits in Porto Conte Ricerche laboratory where I had a pair of these cameras available, so the resulting program is capable of receiving imagery from this type of cameras.

6.2 Implementation details

6.2.1 Choosing the correct algorithm for disparity map

Initial experiments were about choosing the best algorithm for disparity map reconstruction. I decided to stick with the OpenCV methods. Block matching (4.1) method didn't produce satisfying results, so this one was out. At first I thought that Graph-cut (4.3) method will be the best for this purpose. Resulting disparity map had very good quality, however the time consumed by this method was enormous, therefore unsuitable during development. Then I focused on Semi-global Block matching (4.2) method as I was using it during development to save time. During the time I managed to tune the method's parameters a bit to produce better results and I also figured out some postprocessing methods as described in 4.5, which improved the resulting disparity map quality. The resulting disparity map's quality showed to be enough good so I decided to use *Semi-global Block matching* algorithm.

6.2.2 Choosing the most suitable camera setup

During development, I was trying first parallel and then converging setups, Although parallel setup seems to be simpler and less problem-prone, I still had problems in disparity map quality with this one. When I switched to converging setup, disparity map quality ameliorated. I then equipped my program with targetting circle in the middle of the images to be able to estimate the camera placement better.

6.3 Testing

Test set was mainly myself during development as I was in different light conditions, used different camera setups and used different settings of the disparity map reconstruction method and postprocessing. Not every model pans out, mainly because of illumination and automatic exposure and white balance of the cameras. And even though that my webcams have the same resolution, the model quality cannot beat the quality from the GE680 cameras.

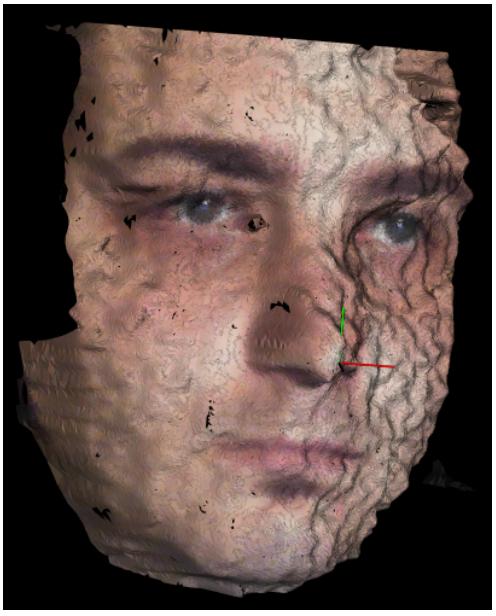


Figure 6.1: Webcam model, render using PCL.

Figure 6.2: Webcam model, render using manual method (same model as before).

Figure 6.3: Webcam model with errors in disparity map, manual method.

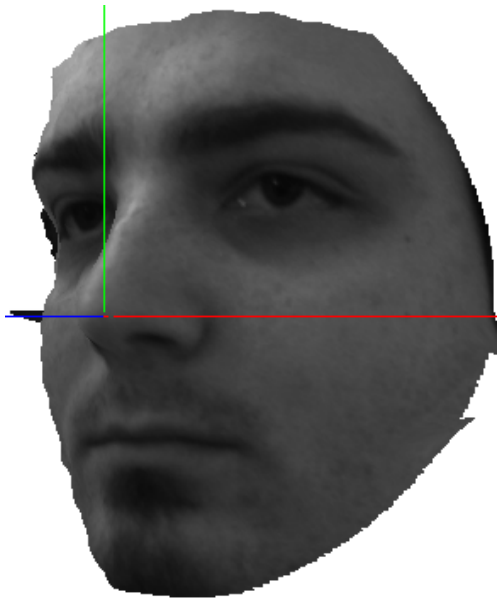


Figure 6.4: GE680 model, manual method.

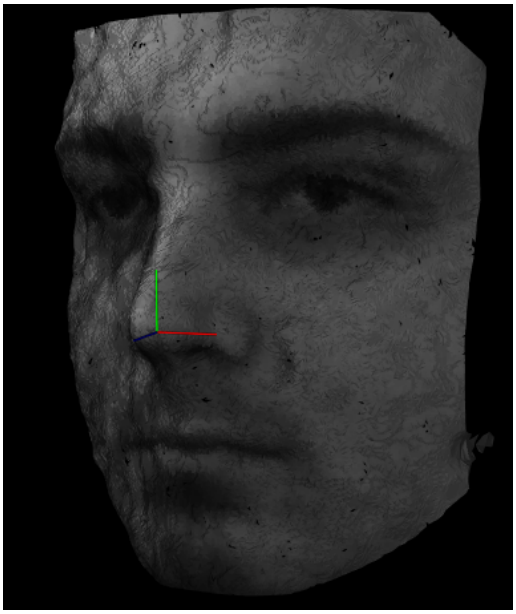


Figure 6.5: GE680 model, PCL method (same model as before).

Chapter 7

Conclusion

The purpose of this thesis was to get familiar with techniques used for computer vision and design a system which could make a model of human face through stereo webcam system. The output is a program which can guide user through the process of calibration and reconstruction.

All knowledge used in this thesis was completely new for me as I never worked with graphics before. I familiarized myself with OpenCV library, which is often used, multi-platform library for computer vision. I discovered how does the calibration and reconstruction work in stereo system. I built a program which tries to make this process as much easy as possible. Complete program is designed as a Qt widget, therefore possibly usable in another applications (after minor modifications - adding signals and slots with communication with other widgets)

I also discovered during work on this thesis that another methods exist, even with only one camera, however this solution is probably the easiest to implement as all parts already exist. There are, however, some drawbacks as this method relies solely on disparity map.

For future development, I would like to improve reconstruction quality by not relying solely on the disparity map by using some kind of biometric approach, like having separate reconstruction procedures, each specializing on particular part of face, together with elimination of artifacts, which are created by hair etc.

Bibliography

[1] Point cloud library. `http://pointclouds.org/`.

[2] Parallel vs. converged. `http://goo.gl/uzcMUV`, 2011 [cit. 2013-07-14].

[3] Semi-global block-matching algorithm. `http://goo.gl/kCepqP`, 2012 [cit. 2013-07-13].

[4] G. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly Media, 2008. ISBN 978-0-596-15620-6.

[5] Cheng-Wei Chou, Jang-Jer Tsai, Hsueh-Ming Hang, and Hung-Chih Lin. A fast graph cut algorithm for disparity estimation. In *Picture Coding Symposium (PCS), 2010*, pages 326–329, 2010.

[6] Sébastien Roy; Ingemar J. Cox. *A Maximum-Flow Formulation of the N-camera Stereo Correspondence Problem*. NEC Research Institute, 2001. ISBN 80-7302-0007-6.

[7] O. D. Faugeras and G. Toscani. The Calibration Problem for Stereo. In *Proceedings, CVPR '86 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, June 22–26, 1986)*, IEEE Publ.86CH2290-5, pages 15–20. IEEE, 1986.

[8] Olivier Faugeras. *Three-Dimensional Computer Vision - A Geometric Viewpoint*. The MIT Press - Massachusetts Institute of Technology, 1993. ISBN 0-262-06158-9.

[9] Berthold K. Horn. *Robot Vision*. McGraw-Hill Higher Education, 1st edition, 1986.

[10] J. Karathanasis, D. Kalivas, and J. Vlontzos. Disparity estimation using block matching and dynamic programming. In *Electronics, Circuits, and Systems, 1996. ICECS '96., Proceedings of the Third IEEE International Conference on*, volume 2, pages 728–731 vol.2, 1996.

[11] Robert J. Schalkoff. *Digital Image Processing and Computer Vision*. John Wiley and Sons, 1989. ISBN 0-471-85718-1.

[12] Frank Y. Shih and Chao-Fa Chuang. Automatic extraction of head and face boundaries and facial features. *Inf. Sci.*, pages 117–130, 2004.

[13] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.

Appendix A

CD contents

- Source code of the program (folder *src-prg/*)
- LaTeXsource code of this thesis (folder *src-doc/*)
- Sample models in XML format (folder *sample-models/*)

Appendix B

Manuals

This chapter shows how to compile and use the attached program.

B.1 Compilation

For compilation, one needs these prerequisities:

- Qt - at least version 4.8, version 5.0 should work too. Lower version of 4.x series Qt might also work as the program does not use the very special features of Qt toolkit.
- Qt Creator - It is possible to compile the program without Qt Creator (by using `qmake` command, see below), but this is more convenient.
- GCC (C++) - versions 4.6 and up.
- OpenCV - version 2.4.5. Higher version should do fine.
- PCL - version 1.6.0
- PvApi - version 1.26

First of all, you have to have all prerequisities installed. Then it's needed to open the `3DReconstruction.pro` file to modify the paths to the libraries (`INCLUDEPATH` and `-L/...` in `LIBS`). You can skip this step if you have Fedora 18 system installed (and possibly higher versions). There is one little glitch with PvApi. It ships with libraries only for GCC 4.1-4.5. However there is no problem with using the 4.5 library in 4.7.

If you want to compile and run the program, easiest way to do so is by using the Qt Creator as mentioned before. Just open the `3DReconstruction.pro` file in Qt Creator and then use its facilities to compile, build and run the program. Classic way is also possible - use the command `qmake -o Makefile 3DReconstruction.pro` (`qmake` can have different name in your operating system, like `qmake-qt4` etc. Check this in your operating system) to generate the Makefile and then use the `make` command to build the executable.

B.2 Usage

Figure B.1 shows the main program window. It's divided into three parts - left camera, right camera, toolbox. Images of the cameras are shown on places marked **B**. Camera selectors are in the places marked **A**. Selection is done in this way:

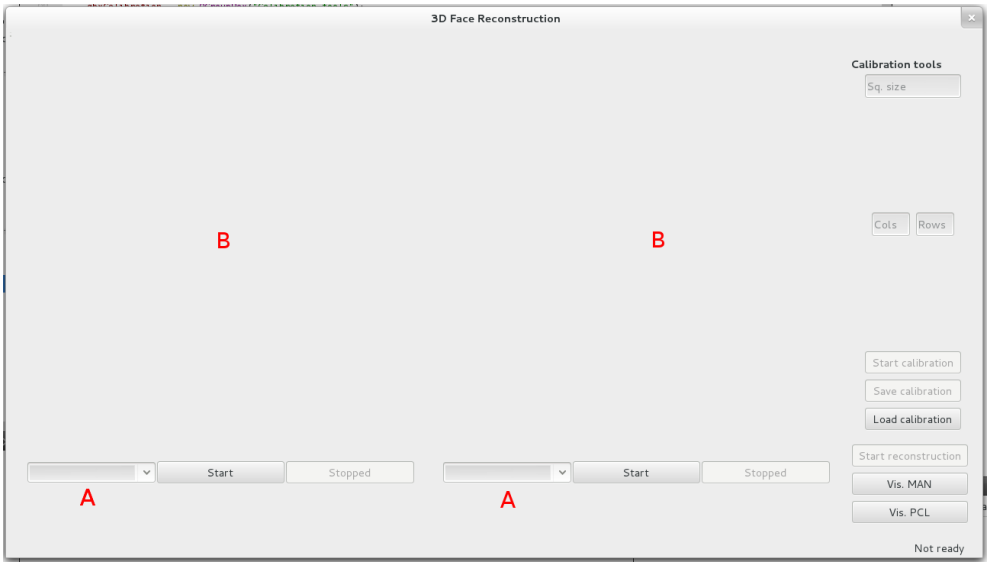


Figure B.1: Program window

- You have GigE cameras. Then wait a moment and the selectors will be populated with all reachable GigE cameras in format `GigE <unique id>`.
- You have a common webcam or another capture device that can be recognized by OpenCV's interface. Select it by typing `webcam <id>` into the selector, where `<id>` is an integer number representing device. Numbering is usually sequential, so first plugged webcam is 0, second 1 and so.

If you have an OpenCV version with built-in GigE support, it's possible to access the GigE cameras through OpenCV's API too, but not by their unique id's (which can possibly swap the cameras).

If you start the feed from cameras, you will see normal picture. You can load previously stored calibration by clicking on *Load calibration*. After loading the calibration, you will now see pair of rectified images. To start a reconstruction, click on *Start reconstruction* whilst looking on an imaginary point between cameras. Then a save file dialog appears. Select folder where you want to place the resulting model and name it. Without an extension, please. It will save two files - disparity map (`.png`) and the model itself (`.xml`).

To run the calibration, one needs a calibration checkerboard (C). Enter the length of the each square's edge into the field stating *Sq. size*, number of rows and cols in appropriate fields. Then start the calibration process by clicking on *Start calibration*. The same button (now saying *Stop calibration*) stops the process. During the process, the program tries to capture the checkerboard every 5 seconds. It informs you about the number of captured frames in an information text in both images. Try capturing the checkerboard in various positions, at least in 15 images. After stopping the process, application freezes for a while (depends of image count) and after that a message box will appear which will tell you how good is the calibration. Then you can save the calibration by clicking on *Save calibration* button.

Buttons *Vis. MAN* and *Vis. PCL* serve for visualizing existing models. MAN means manual method, PCL means it's reconstructed by using PCL library as described in chapter 5.

I must concede that there are two places where a segmentation fault can come from, both from the libraries. When you close the program during the time when PvAPI does something (e.g. scans network for cameras, etc.) and sometimes PCL visualization method does this too when points are inserted into search tree for normal estimation.

Appendix C

Calibration checkerboard

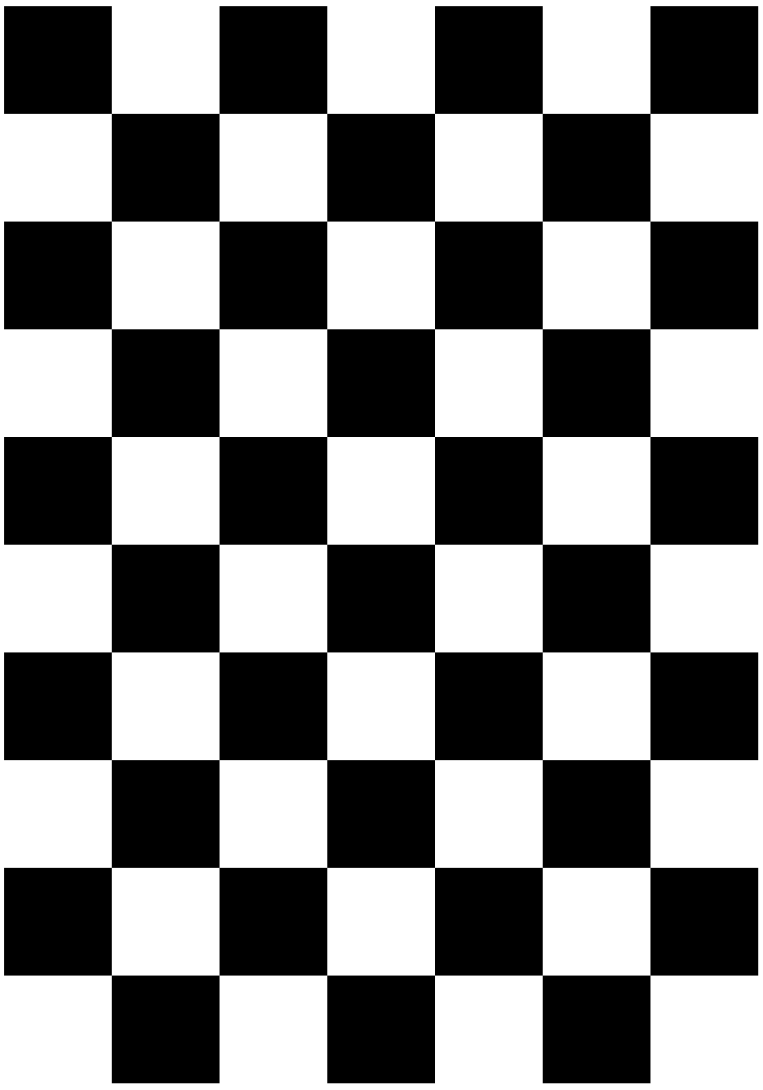


Figure C.1: 9x6 calibration checkerboard. Length of each edge is 1.43 cm