Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Hedvika Peroutková

Extrakce fototextur pro 3D fotometrickou rekonstrukci

Phototextures extraction for 3D photometric reconstruction

Kabinet software a výuky informatiky
Vedoucí diplomové práce: Ing. Jan Buriánek
Studijní program: Informatika, Softwarové systémy

2008

First of all I'd like to thank Ing. Jan Buriánek for his assistance during my work on this thesis.

I'd like to thank Visual Connection, a.s. and The Museum of the City of Prague for the opportunity to participate in a unique digitization project and for providing data and pictures for my work, especially to PhDr. Zuzana Strnadová, Mgr. Pavla Státníková, PhDr. Kateřina Bečková, Prof. Ing. Jiří Žára CSc. (external advisor from ČVUT FEL) from The Museum of the City of Prague and Bc. Tomáš Petrů, Ing. Jakub Vaněk, Jakub Kabourek and Ing. Tomáš Pavlis from Visual Connection, a.s.

I'd also like to thank my friends and colleagues Ing. David Sedláček, Ing. Jan Kirschner, Matěj Cáha and Lukáš Mach for their invaluable help and advice. I'd like to thank my family for their love, understanding and encouragement during my studies and work on this thesis.

Prohlašuji, že jsem svou diplomovou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 6. 8. 2008 ....

Contents

1 Introduction **1**

2 Problem of texturing **3**

2.1 Input data 3

3 Analysis and solution **5**

3.1 Extraction 5

3.1.1 Extraction using z-buffer 5

3.1.2 Extraction using raycasting 7

3.1.3 Extraction using border texture points 8

3.2 Registration algorithm classification 10

3.3 Feature-based registration 10

3.3.1 Harris corner detector 10

3.3.2 SIFT 12

3.4 Registration via deformation models 15

3.4.1 Deformation model definition 15

3.4.2 Algorithm design 17

3.4.3 TRW-S 18

3.4.4 Method of finding incomplete min-marginals in $\mathcal{O}(\mathcal{L})$ 19

4 Realization **22**

4.1 Platform choice 22

4.2 Project structure 22

4.3 Feature method 23

4.3.1 Texture acquisition 23

4.3.2 Warping textures 23

4.3.3 Building final textures 26

4.4 Deformation model method 26

4.4.1 Warping textures 27

4.4.2 Building final textures 28

4.5 Usage of developed programs 28

5 Digitalization of Langweil's model of Prague **31**

5.1 Model specifications 31

5.2 Workflow 32

6 Testing and method comparison **35**
6.1 Deformation model comparison 35
6.2 Feature method versus deformation model method 37

7 Conclusion and future work **40**

A Configuration file example **43**

B Abbreviation List **44**

C Contents of CD **45**

List of Figures

2.1 On the left image we can see hardly visible textures in the narrow streets, the right image shows texture affected by small depth of field. The top of the texture appears sharp, the bottom is blurred [1]. . . . 4

3.1 Rasterization vectors x_r and y_r are given by the ratio between size of the face in 3D and size of the 2D bitmap. 6
3.2 Casting rays from the center of projection to points on the texture. . 7
3.3 Perspective camera projection. 9
3.4 Harris response to two textures of the same roof from different cameras. For white areas maximum values are found in both images and then matched using image information. 11
3.5 Finding extremes in scale-space of difference of Gaussians [11]. 13
3.6 Scale-space of difference of Gaussians [11]. 14
3.7 Gradient array and SIFT descriptor visualization [11]. 15
3.8 Penalization function derived from discontinuity. 16
3.9 (a) Two neighboring blocks of 4×4 pixels; (b)-(d) Examples of block displacements which would be only "soft" penalized. It enables better representation of the image transformation, e.g. scaling [14]. 17
3.10 A grid with $N = 9$ points containing the point set $P = \{1, 3, 7\}$. The cones are v-shaped of slope 1. The distance transform as height of lower envelope, depicted as a dashed contour. The example is reprint from [5]. 19
3.11 One-color functions $\theta^i_{st}(x_s, x_t)$ shifted vertically by the value $\Psi^i_t(x_s) + \theta^i_t(x_s)$, where x_s is $x_1 ... x_6$. The minimum of all these functions at each location is a lower envelope, depicted as bold contour. 21

4.1 Workflow of texture processing with two different data inputs. 23
4.2 Example of the textures with marked border points and edges [1]. In the top left corner the chimney is also marked as a salient object for a higher quality of warping process. 24
4.3 Color lines representing SIFT correspondences in two different images filtered in area by a grid. 25
4.4 Texture built from different views around chimney [1]. 26

5.1 View over the model. Incredible details on the model [1]. 31
5.2 Point cloud of the entire Langweil's model [1]. 32

5.3 Dots on the image show systematical image capturing in a grid with 3 different camera elevations α and 8 different azimuths ω. We can see 3 different focus planes π^1, π^2, π^3 which correspond to three camera heights above the model. Courtesy of Jan Kirschner [9]. 33
5.4 Our workflow of digitalization. Courtesy of Matěj Cáha [4]. 34

6.1 The reference image is on the left, the image deformed by program TextureWarper is in the middle and target image is on the right side [1]. 36
6.2 The reference image is on the left, the deformed image created by program TextureWarper is in the middle and image on the right side was deformed by program DeformMatch [1]. 37
6.3 Local imperfection in texture caused by mismatching feature points [1]. 38
6.4 The neighboring textures don't pass together, their structure and brightnesses differ and the texture on the right involves parts of other faces [1]. 38

Název práce: Extrakce fototextur pro 3D fotometrickou rekonstrukci
Autor: Hedvika Peroutková
Katedra: Kabinet software a výuky informatiky
Vedoucí diplomové práce: Ing. Jan Buriánek
E-mail vedoucího: jan.burianek@visual.cz

Abstrakt: Tato práce se zabývá problémem získávání textur z fotografií, zejména pro počítačové modely budov. V práci je využívána skutečnost, že textura je snímána z více úhlů a jsou známy korespondující hraniční body. V rámci této práce byl vytvořen systém pro extrahování textur z fotografií, hledání korespondencí v texturách a spojování textur nasnímaných z různých úhlů do co nejkvalitnější výsledné textury vhodné pro daný model. Problém registrace textur získaných z různých snímků je řesen jednak pomocí příznakové metody na bázi SIFT deskriptorů, jednak pomocí deformačních modelů. Implementované metody jsou použitelné i pro texturaci rozsáhlých modelů v dostatečné kvalitě. Úlohu omezujeme na texturaci modelů po částech složených z ploch.

Klíčová slova: extrakce textur, registrace obrazu, 3D rekonstrukce

Title: Phototextures extraction for 3D photometric reconstruction
Author: Hedvika Peroutková
Department: Department of Software and Computer Science Education
Supervisor: Ing. Jan Buriánek
Supervisor's e-mail address: jan.burianek@visual.cz

Abstract: This thesis deals with the problem of creating textures from photographs, especially for a computer model of buildings. The fact that images are taken from different views and border points of a texture are known is supposed in this thesis. Within the frame of this thesis a system for texture extraction from photographs, searching correspondences in textures and merging textures from different views to the final high quality texture suitable for a given model was developed. The problem of registration textures obtained from different images is solved both by feature method based on SIFT descriptors and by method arisen from a deformation model. The implemented methods are also applicable for texturing of huge models in sufficient quality. We restrict our task to texturing 3D surface models built up from planar faces.

Keywords: texture extraction, image registration, 3D reconstruction

Chapter 1

Introduction

The task of extracting textures for real objects from photographs is one of the key problems in the process of 3D reconstruction. As there is a natural trend of creating 3D models of objects, buildings, places around us, a need for automated 3D reconstruction process becomes more significant. The quality of texture extraction and registration is the main factor that affects the quality of the resulting 3D reconstruction. Textures from different images need to be enhanced and the fact that we can't assume that camera parameters, lighting conditions and 3D model geometry are exactly known makes this task difficult.

The solution of this problem can be used in 3D reconstruction and other fields of computer vision as well as in industry.

Nowadays, systems dealing with this problem are for instance the program ImageModeler developed by Realviz, program Canoma from Adobe or PhotoModeler developed by Eos Systems. However, results from the existing solutions are satisfying only for simple models that can be reconstructed from small set of photographs. When a large number of images is used, programs are not stable or are not able to use information from images obtained from different views, especially when textures are partly occluded.

This thesis deals with the problem of texture creation for a 3D model, where the goal is to create the best texture for each model face from given images. This means that the texture should be in high resolution, sharp enough and should cover all texture parts which can be seen from at least one camera. To meet all these requirements, it is necessary to create complex utility for image matching, which could help to prepare textures by exploiting all given information.

The goal of this thesis is not a complete model reconstruction, but only texture creation for already reconstructed model. As the reconstruction of model vertices and their back projection is very sensitive to the precise estimation of the camera calibration parameters and the accuracy of reconstructed point positions, one part of this thesis used also information about border texture points on images. By involving this information we can improve the texture accuracy but it is also limited by the fact that borders are generated manually. This process can be used only for small or not very complex models.

In chapter 2, various aspects and problems of texturing are discussed. Various methods for texture extraction and image matching are surveyed in chapter 3. Chapter 4 describes the applied methods that are suitable for our problem. Proposed solutions were used and tested in the project of digitalization of Langweil's model of Prague which due to its size enables to thoroughly test our algorithms, this is further described in chapter 5. In chapter 6, our solutions are compared to the existing ones. Chapter 7 concludes our results and offers possible improvements for the future.

Chapter 2

Problem of texturing

The problem of model texturing can be divided into three steps. It is texture extraction from photographs, image matching of extracted textures and texture building for attaining final texture from deformed textures. Texture extraction can be realized by means of ray casting if we had a 3D model of scene or using information about texture borders. In the second case it is easy to implement but there is need to establish border points or edges by hand, in the first case we don't need border information, however textures can be affected by reprojection error.

Image matching is the key step in the texturing process. Final quality of textures particularly depends on efficiency of matching system. There are two main possibilities how we can estimate a mapping from one image to another. Firstly, it can be realized by finding features in both images, matching them and warping image piecewise. Secondly, we can compute deformation field for whole image pixel-wise (or block-wise).

Building the final texture should be done with respect to the picture illumination, local texture sharpness and size of not occluded parts of texture. It means textures should be bright balanced before using, only focused parts of texture without reflections should be used and they should be added to the final texture according to size of their contribution to the final texture.

2.1 Input data

The quality of final textures greatly depends on the quality of images of reconstructed object. For attaining textures of sufficient quality, we need enough images from different angles, especially if we prepare textures for some structured object. On the other hand, a large number of images assumes bigger demands to our system. Anyway we usually don't have many different images for faces which are not at the easily visible places, in some small or narrow places of the object. Hence most cameras show only small parts of the texture or they show textures from very small angle.

Another aspect that should be considered is texture blur caused by camera's focus and depth of field. We can't expect to get textures that are sufficiently sharp

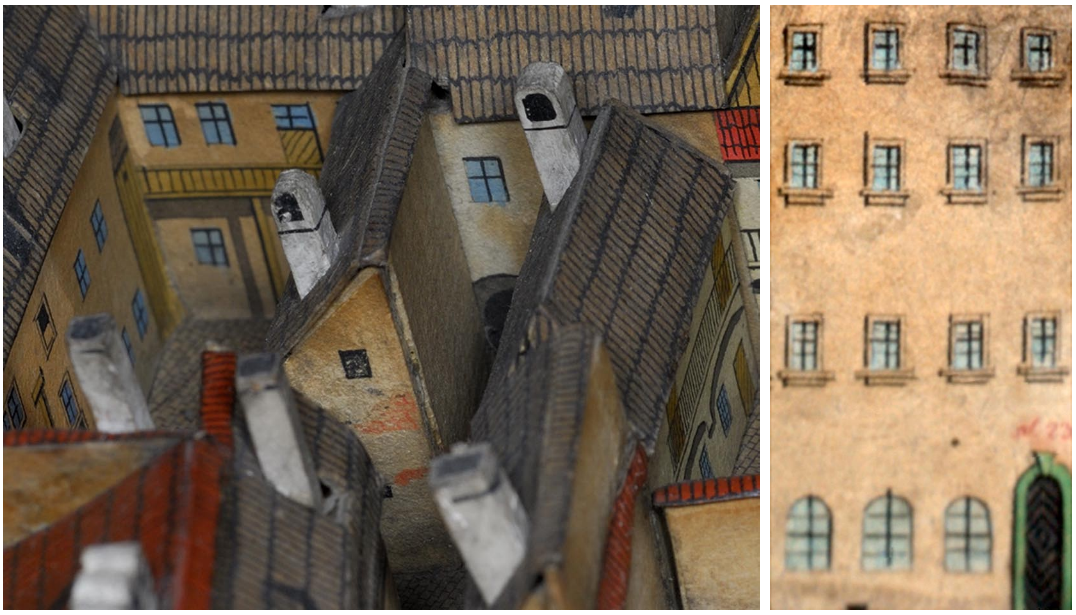
Figure 2.1: On the left image we can see hardly visible textures in the narrow streets, the right image shows texture affected by small depth of field. The top of the texture appears sharp, the bottom is blurred [1].

over their whole area, especially if we take photos with small depth of field.

We must also consider the surface of textured objects before texturing process. Reflections from the surface affects negatively color of scanning object. If we use a flash within capturing images of our scene, we should expect unpleasant surface reflections. These damaged parts of textures should be detect and not used for the final texture building. Recently, Jan Kirschner deals with this problem in his thesis[9].

Texturing is usually done for wired models of objects. Surface of this model is built up from plains, or mathematically easy defined surfaces. It is simplification of real bumpy surfaces, which is necessary for reconstruction; however, it may cause problems within the texturing process. Image registration system should be able to process textures extracted from not exactly planar surfaces, which causes deformations in the texture viewed from different angles.

Chapter 3

Analysis and solution

3.1 Extraction

The texture extraction is the first step in the texturing process. It can be performed using various methods depending on different types of input data. As texturing isn't usually performed without previous processing of the model reconstruction, input data are given from this process.

We typically know the 3D model geometry and camera calibrations and we have to extract textures using this information. In this case, we are also able to extract information about the texture visibility over its area, the angle between a face normal and a ray from camera to each pixel and also distances of texture pixels from the focus plain within capturing the image, supposing that we know used camera's focal length. There are two main methods how we can achieve this. These methods are based on z-buffer or on raycasting. For computation of the complexity we assumed that we have camera count N_C with resolution R_C and face count N_F with common face resolution R_F.

3.1.1 Extraction using z-buffer

Using this method we can extract all textures from one image at once. For each image we allocate a buffer (a two-dimensional array) with resolution of our images containing information about the front face for each pixel. The texture visibility is saved directly in z-buffer. However, we have to look for each texture surroundings and from depth information identify at texture boundaries possible occlusion of farther texture by closer one. Possible occlusion should be considered because the model geometry and camera calibration aren't usually absolutely exact, moreover, small depth of field can cause that the front blurred face affects the texture of farther face. Therefore unpleasant artifacts could appear in the extracted textures. The observation angle of each visible texture can be computed from the model geometry. Local texture sharpness is obtained for rasterized texture from 3D data. Each texture should be extracted by means of projection of its face in the given camera to the area corresponding to perpendicular view of our texture. This can lead to

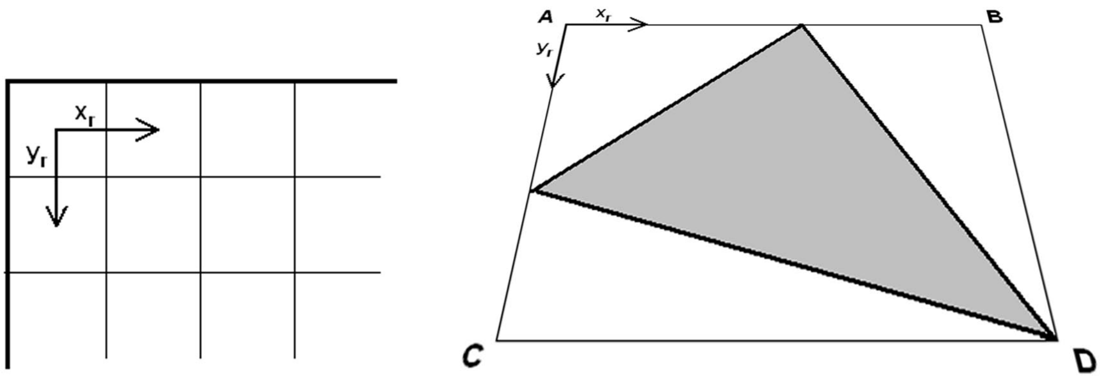

Figure 3.1: Rasterization vectors x_r and y_r are given by the ratio between size of the face in 3D and size of the 2D bitmap.

loss of texture information if the target texture size in perpendicular view isn't large enough. The z-buffer preparation has a computational complexity of $\mathcal{O}(N_C N_F R_C)$, texture transformation to perpendicular view $\mathcal{O}(N_C N_F R_F)$, estimation of possible occlusion can be also computed in $\mathcal{O}(N_C N_F R_F)$ using only information at the texture boundaries. Texture sharpness has the same complexity and observation angle of the texture can be estimated either for whole texture in $\mathcal{O}(N_C N_F)$ or more exactly for each texture pixel in $\mathcal{O}(N_C N_F R_F)$. Total complexity of this z-buffer-based method is therefore $\mathcal{O}(N_C N_F R_C + k N_C N_F R_F)$, where $k = 3$ or 4 depending on the chosen observation angle estimation method, which leads to $\mathcal{O}(N_C N_F R_C)$.

The part of this method is texture rasterization process. It is the transformation of a 3D face to a 2D bitmap by transformation a 2D base of a texture bitmap to a 2D base of the texture face in coordinate system of the model. The rasterization function is a function which assigns to a point $x \in \mathbb{R}^2$ a point $x' \in \mathbb{R}^3$, $x' = raster(x)$. The rasterization function can be expressed via the minimal point A and two rasterization vectors x_r, y_r.

$$raster(x) = A + x_1 \cdot x_r + y_1 \cdot y_r,$$
$$x = (x_1, y_1),\ A, x_r, y_r \in \mathbb{R}^3$$

Rasterization vectors can be seen on the Fig. 3.1. If A, B, C and D are vertices of circumscribed rectangle of the face (rotation of the rectangle in the face plain is given by the face rotation in final texture bitmap) and $hres$, $vres$ are horizontal and vertical resolutions of texture in bitmap, then x_r, y_r are given by the expression:

$$x_r = (B - A)/hres$$
$$y_r = (C - A)/vres$$

The minimal point A and rasterization vectors x_r, y_r create the base of the texture. However, textures are usually polygons not only triangles and their points usually don't lay in one plain, which can be caused by an inaccuracy of the model or the item itself can be deformed in various ways. In this case, polygons are triangulated (e.g. by Delaunay triangulation) and approximated by a plain and the triangles of this polygon can be projected to this plain and rasterized as a planar polygon.

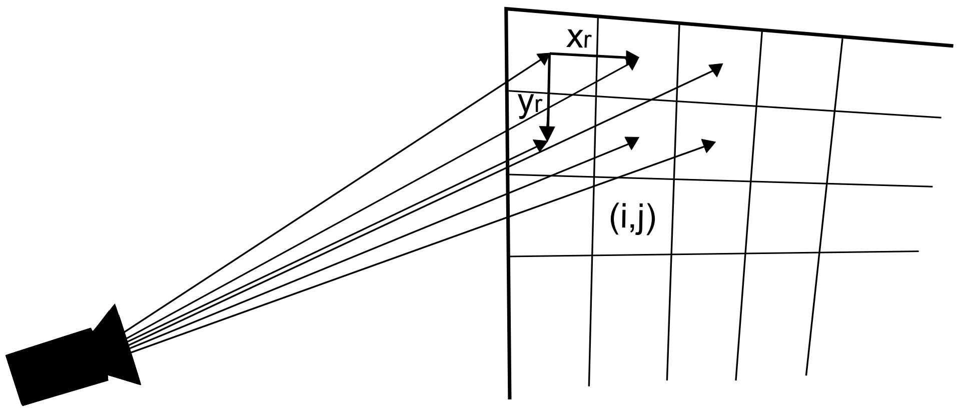

Figure 3.2: Casting rays from the center of projection to points on the texture.

3.1.2 Extraction using raycasting

The method based on raycasting extracts a texture from the camera by casting a ray from each texture pixel. It is more difficult method than computing with z-buffer, but it uses model information in better way, such as relative positions of faces. A ray is casted from the texture pixel and it is necessary to detect collisions with other faces within traversing the model. For enabling the estimation of a possible occlusion in the given pixel, we have to adjust either the faces to bigger ones with partly transiting margins or to traverse the model with a cylinder instead of a ray. We have to extract $N_C N_F$ textures with R_F pixels and for a ray casted from each pixel, we have to compute collisions with other $N_F - 1$ faces. For speeding up the traversing, we can use some accelerating structure, such as k-d tree. It gives us complexity $\mathcal{O}(log(N_F))$, which leads to the total complexity $\mathcal{O}(N_C N_F log(N_F) R_F)$.

Rays are casted from camera position to the points of rasterized faces representing pixels of extracted textures. When we extract texture pixel (i, j) from camera C with position C_{pos}, where A is the minimal face point and x_r, y_r are rasterization vectors of the texture, then the ray for traversing the model is defined as:

$$R = C_{pos} + t \cdot d_r$$
$$d_r = A + x_r \cdot i + y_r \cdot j - C_{pos},$$

where d_r is direction of ray. Polygons can be easily extracted by extracting all of their triangles and composing the whole texture. By means of kd-tree collisions with other faces are found and ray R can be with actually extracted triangle T in three different position.

1. R and T don't have an intersection, extracting pixel is out of the triangle.

2. R and T have an intersection, but the distance from C_{pos} is greater than the distance of intersection with any other face. Then the pixel of triangle T is occluded.

3. R and T have an intersection, which is the nearest one. The pixel color for the texture can be retrieved from the camera C.

In the third case we can get pixel color for texture using the information about 3D point $I = A + x_r \cdot i + y_r \cdot j$ and camera projection matrix. Image coordinates $x/w, y/w$ are computed using expression:

$$w \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{P}^{3\times4} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

where $\mathbf{P}^{3\times4}$ is projection matrix of camera C. Projection matrix is composed from calibration matrix $K^{3\times3}$, rotation matrix $R^{3\times3}$ and a vector translating center of projection $T^{3\times1}$ according to following formula:

$$P = [KR^T| - KR^TT]$$

The matrices K, R, T can be composed using camera calibration information.

$$K = \begin{pmatrix} -f_c \cdot k_x & s & x_0 \\ 0 & f_c \cdot k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$R = \begin{pmatrix} \cos\alpha_z & -\sin\alpha_z & 0 \\ \sin\alpha_z & \cos\alpha_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha_y & 0 & \sin\alpha_y \\ 0 & 1 & 0 \\ -\sin\alpha_y & 0 & \cos\alpha_y \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_x & -\sin\alpha_x \\ 0 & \sin\alpha_x & \cos\alpha_x \end{pmatrix}$$
$$T = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix},$$

where x_0, y_0 are coordinates of the principal point, $x_0 = res_x/2$, $y_0 = res_y/2$, s is skewness factor, k_x, k_y are focal lengths along the x and y axes, which can be expressed as:

$$k_x = \frac{res_x}{2f_c \cdot \tan\frac{fov_x}{2}}$$

After computing the real values x, y we can get image coordinates using nearest neighbor interpolation. We use a pixel color from the image for our texture pixel (i, j). Pixel focusing over the texture can be estimated from 3D geometry of the face and camera calibration parameters. The observation angle of the texture can be estimated either for whole texture or pixel-wise as in the previous case.

3.1.3 Extraction using border texture points

We can't always suppose that we have a prepared 3D model geometry before the start of texturing process. It might happen that the reprojection error is considerably

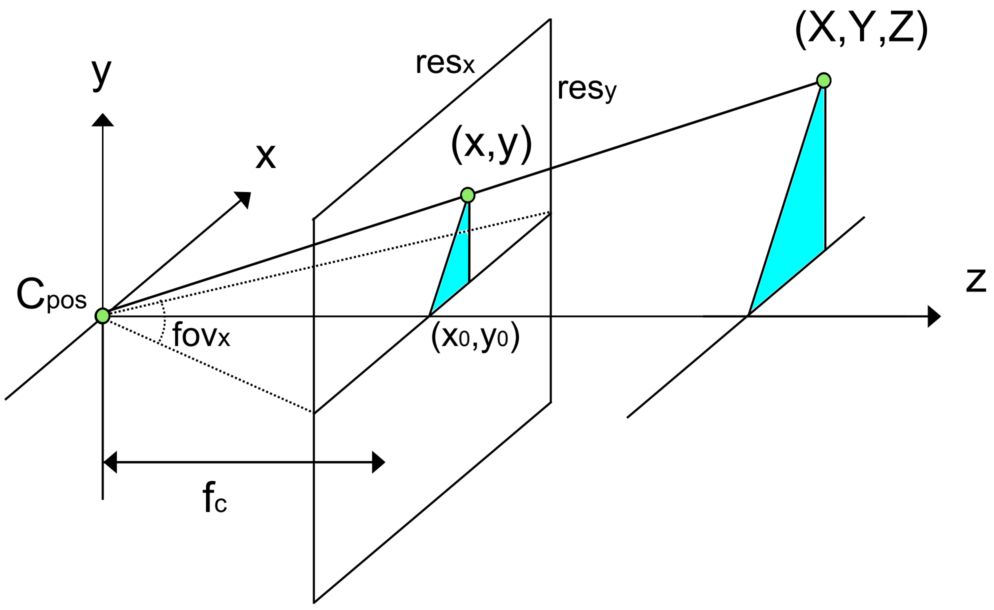

Figure 3.3: Perspective camera projection.

high and it is necessary to add 2D information from images about border points for accuracy improvement or we obtain 2D point correspondences which a 3D model is then prepared from. In this case we have no information about face positions in the 3D space, not even the distances of face border points from the camera, which we could use for the texture rasterization.

By simple cutting these faces from images, we get a set of textures of various sizes and various perspective deformations. For an approximate texture fitting we can compute homography matrix from their border points. The homography is defined in 2D space as a mapping between the point on the plane as seen from one camera, to the same point on the plane as seen from another camera.

$$p'_b = H_{ab} \cdot p_a$$

$$p'_b = \begin{pmatrix} x_b \cdot \omega' \\ y_b \cdot \omega' \\ \omega' \end{pmatrix}, \; p_a = \begin{pmatrix} x_a \\ y_a \\ 1 \end{pmatrix}, \; H_{ab} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix},$$

where p_a, p_b are homogeneous coordinates of 2D border points. Usually a more appropriate model of image displacements is used, which is called affine homography. The affine homography is a special type of a general homography, whose last row is fixed to $h_{31} = h_{32} = 0$ and $h_{33} = 1$. For computing the homography we need at least 4 pairs of corresponding non-collinear points. In the case of affine homography, 3 points are needed. Details about the estimation of homography can be found in [8].

The disadvantage of this approach is that the homography can be used only for planar or almost-planar textures. Also some image information can be lost when we use homography. Hence, it has to be considered whether our registration methods are enough robust to scale changes, or whether we want to use homography and acquire approximate range of correspondence point positions within the texture borders.

After extraction process is finished, wide-baseline stereo problem must be solved. It is problem of establishing correspondences between two textures taken from different viewpoints. The original image, which is kept unchanged, will be referred to as the reference image and the image to be mapped onto the reference image will be referred to as the target image.

3.2 Registration algorithm classification

Image registration algorithms fall within two sections of classification: area-based methods and feature-based methods. Feature-based methods detect salient points in the images and the mapping from one image to another is set according to the correspondences of these image features (points, lines, curves, closed-boundary regions, etc.) Alternatively for area-based image registration methods, the algorithms don't attempt to detect salient objects, but they look at the structure of the image via correlation metrics, Fourier properties or by other means of structural analysis. Usually windows of predefined size or entire images are used for the correspondence estimation. The special case of area-based method is registration based on deformation model. A well-arranged review of existing image registration methods can be found in [15].

3.3 Feature-based registration

This approach is based on searching salient structures - features in the image. Points (like line intersections, region corners, salient points on curves), lines (like region boundaries) or regions are understood as features. These features should be distinctive, spread over the whole images, repeatable (detectable in both registrated images in the sufficient amount) and accurately localized (with pixel or subpixel position). Comparison of different feature-based method can be found in [6].

One of the frequently used features are corners. Harris corner detector is popular interest point detector due to its strong invariance to rotation, scale, illumination variation and image noise.

3.3.1 Harris corner detector

Harris corner detector [7][13] is based on the local auto-correlation function of image intensity I. The local image intensity changes are measured with patches shifted by a small amount in different directions. Our interest points have high changes in every shift. Change of intensity can be computed as the weighted sum of square difference between two patches, one is the window $W(x, y)$ and second is shifted by $(\Delta x, \Delta y)$.

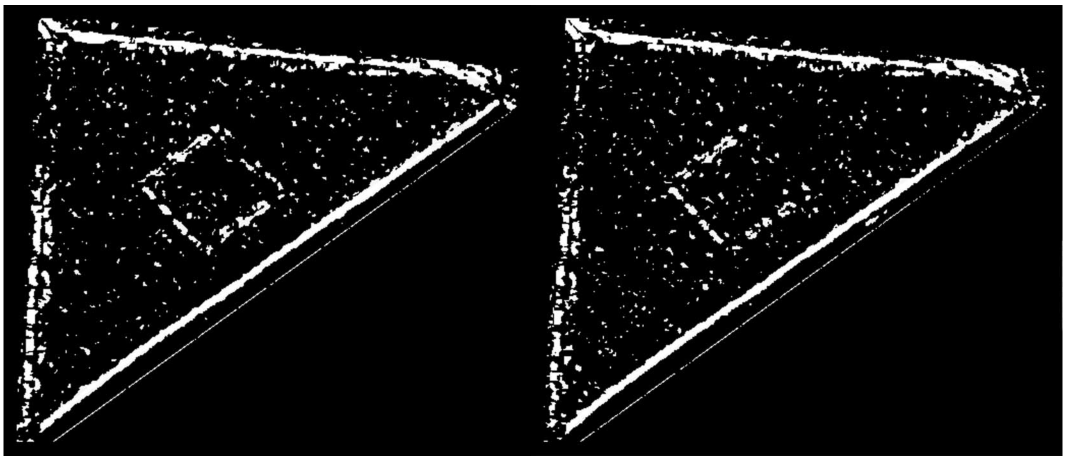
Figure 3.4: Harris response to two textures of the same roof from different cameras. For white areas maximum values are found in both images and then matched using image information.

$$E(x, y, \Delta x, \Delta y) = \sum_{u,v \in W(x,y)} w(u, v)[I(u + \Delta x, v + \Delta y) - I(u, v)]^2,$$

where $w(u, v)$ is a weight function and $W(x, y)$ is a set of pixels in the window centered at point (x, y). When the Harris matrix A is found by approximating S with a second order Taylor series expansion E can be expressed as:

$$E(x, y) \approx \frac{1}{2} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} A(x, y) \, (\Delta x, \ \Delta y).$$

where A is a 2x2 Hessian matrix of second derivatives of image intensity I:

$$A(x, y) = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix},$$

A circular window or circularly weighted window, such as a Gaussian, should be used so that the response can be isotropic.

A corner is characterized by a large variation of E in all directions. This characterization can be expressed via eigenvalues of A. In the interest point A should have two large eigenvalues. There are three cases to be considered:

1. If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ then there are no points of interest at this patch, a windowed image region is of approximately constant intensity.

2. If $\lambda_1 \approx 0$ and λ_2 has some large positive value, local shifts in one direction cause little change in E and significant change in the orthogonal direction. An edge is found.

3. If λ_1 and λ_2 are both large, distinct positive values. Shifts in any direction will result in a significant change of E. A corner is found.

As exact computation of the eigenvalues is computationally expensive, the following function is suggested instead:

$$R = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = detA - \kappa \cdot tr^2(A),$$

where κ is a tunable sensitivity parameter. Therefore, we don't have to compute the eigenvalue decomposition of the matrix A and for finding features it is sufficient to evaluate the determinant and trace of A.

Pixels with R greater than some threshold T are chosen for further processing. The maximum value of regions created by thresholding has to be found. The accurate feature point localization can be done by fitting a parabola to the neighboring pixels of the maximum point.

After feature detection, matching of the obtained features follows. There are three main methods how to perform matching. Firstly, area around feature points from different images can be compared by sum of absolute differences in image intensities. Alternatively, sum of square differences can be used. Another method is correlation of rectangle areas around two feature points. The value of feature correlation can be computed as:

$$C = \frac{\sum\limits_{i \in \{1..n\}} (a_i - \overline{a}) \cdot (b_i - \overline{b})}{\sqrt{\sum\limits_{i \in \{1..n\}} (a_i - \overline{a})^2} \cdot \sqrt{\sum\limits_{i \in \{1..n\}} (b_i - \overline{b})^2}},$$

where a_i, b_i are pixels around feature points, n is number of pixels in this area and $\overline{a}, \overline{b}$ are mean values of a_i, b_i:

$$\overline{a} = \frac{\sum_{i \in \{1..n\}} a_i}{n}, \quad \overline{b} = \frac{\sum_{i \in \{1..n\}} b_i}{n}$$

Feature point with the lowest correlation value from the target image is assigned to feature point from reference image.

3.3.2 SIFT

Scale-invariant feature transform is an algorithm for detection and description of local features in images. It is very popular and effective method for image matching but it is also used for general object class recognition. Feature descriptors are invariant to image scale and rotation and they are also robust to changes in illumination and noise.

The algorithm was published by David Lowe[11]. It includes two steps, feature extraction and feature matching. Feature extraction has 4 phases: scale-space extremes detection, keypoint localization, orientation assignment and keypoint descriptor creation. In the first stage of computation potential interest points are searched over

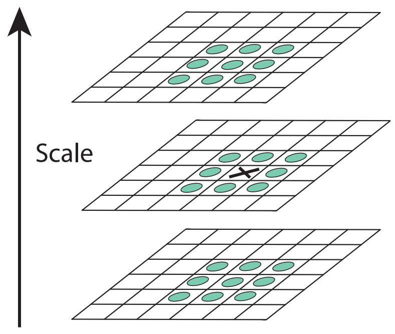

Figure 3.5: Finding extremes in scale-space of difference of Gaussians [11].

all image scales and locations. This is implemented using differences of Gaussian (DoG), see Fig. 3.6.

$$DoG = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y),$$

where $G(x, y, \sigma) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$, I is an input image and $*$ is convolution operation in x and y. Candidate points are local maxima in DoG, see Fig. 3.5. This yields a non-integer position (x, y).

After this first choice by maximizing DoG in scale and in space, accurate keypoint localization follows. Location, scale and ratio of principal curvature have to be found. It is necessary to reject keypoints with low contrast and keypoints lying on the edges, which could be poorly localized. This can be done by fitting a 3-variate quadratic function to surround values of keypoint at selected scaled picture. By finding a maximum (minimum) of this function we get a subpixel localization of keypoint. According to shape of the quadratic function we can reject low contrast keypoints. For eliminating edge responses we can compute 2x2 Hessian matrix and by means of this matrix find out the ratio of its eigenvalues.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

The derivatives can be estimated by differences of neighboring points. In case of edge response DoG has a large principal curvature across the edge but a small one in the perpendicular direction. This can be found out via the eigenvalue ratio of H. The ratio between the larger magnitude eigenvalue and the smaller one r, $\alpha = r\beta$, can be computed as:

$$\frac{TR(H)^2}{Det(H)} = \frac{(r+1)^2}{r}$$

Then checking that ratio is under some threshold decides about preserving or rejecting the keypoint:

$$\frac{TR(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

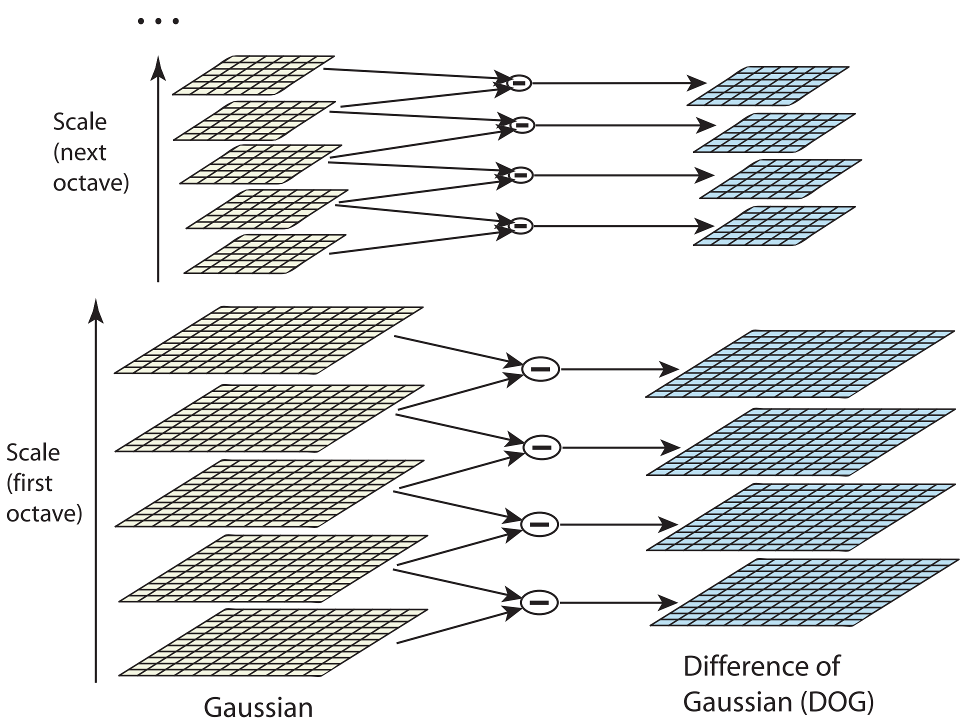

Figure 3.6: Scale-space of difference of Gaussians [11].

where we can use as a threshold e.g. $r = 10$ for eliminating curvature with eigenvalue ratio greater than 10.

To the rest keypoints a local orientation should be assigned. Keypoint descriptor can be then computed with respect to this orientation and therefore it achieves invariance to the image rotation. For image sample $L(x, y)$, let d_x, d_y be local first derivatives in x and y, which can be established by $d_x = L(x+1, y) - L(x-1, y)$, $d_y = L(x, y+1) - L(x, y-1)$. Then gradient magnitude $m(x, y)$ and its orientation $\theta(x, y)$ can be computed as:

$$m(x, y) = \sqrt{d_x^2 \ + \ d_y^2}$$
$$\theta(x, y) = arctan(d_y/d_x).$$

An orientation histogram is created from orientations in sample points within a region 16x16 pixels around the keypoint. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window. Orientations are placed in histogram in 36 bins covering 360 degrees range of orientation. Dominant direction in histogram is detected, and then if any other orientation is presented also at least in 80% frequency of highest direction, another keypoint is created at the same location and scale but different orientation. For finding accurate direction a parabola is fit to the 3 histogram values closest to highest peak in the histogram.

SIFT descriptor is then prepared using sample gradients weighted by their magnitude and Gaussian and rotated relative to the keypoint orientation. They are

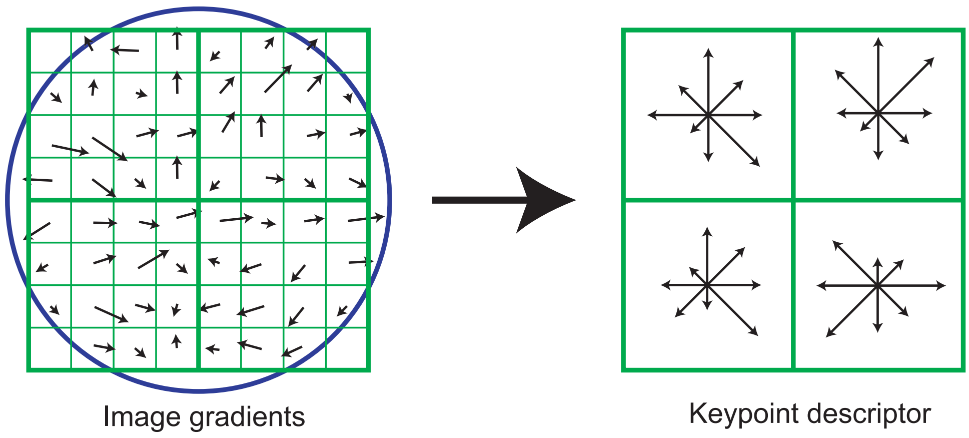

Figure 3.7: Gradient array and SIFT descriptor visualization [11].

divided in 4x4 sample regions and in each of these regions orientation histogram with 8 bins is enumerated. The descriptor is therefore a vector of 128 (8 orientation x 4x4 histogram array) values. This vector is normalized to unit length to make it invariant to affine changes in illumination. As non-linear illumination changes can also occur, the values in the unit feature vector are thresholded and then normalization to unit length should be done again.

The final descriptor with 128 values in $\langle 0, 1 \rangle$ is therefore invariant to image scale and rotation, a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination.

After finding features in both images feature matching is performed. The nearest descriptor from second image is searched for every keypoint. As a distance between descriptor minimum Euclidean distance can be used. For providing an estimate of false matching we should consider also the second-closest match. It is effective to reject all matches in which the distance ratio between the first and the second-closest match is greater than some threshold, e.g. 0.8.

3.4 Registration via deformation models

Deformation models enable finding 2-dimensional discrete mapping from one image to another maximizing image quality. This method is based on Markov random fields.

3.4.1 Deformation model definition

Deformation of reference and target image is represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ antisymmetric and antireflexive. Each graph node s $\in \mathcal{V}$ has an assigned label $x_s \in \mathcal{L}$, where $\mathcal{L} = \{-K...K\}$ is a finite set of labels. Labeling (configuration) is defined as $\mathbf{x} = \{x_s \mid s \in \mathcal{V}\}$. Let $\{\theta_s(i) \in \mathbb{R} \mid i \in \mathcal{L}, s \in \mathcal{V}\}$ be univariate potentials and $\{\theta_{st}(i, j) \in R \mid i, j \in \mathcal{L}, st \in \mathcal{E}\}$ be pairwise potentials.

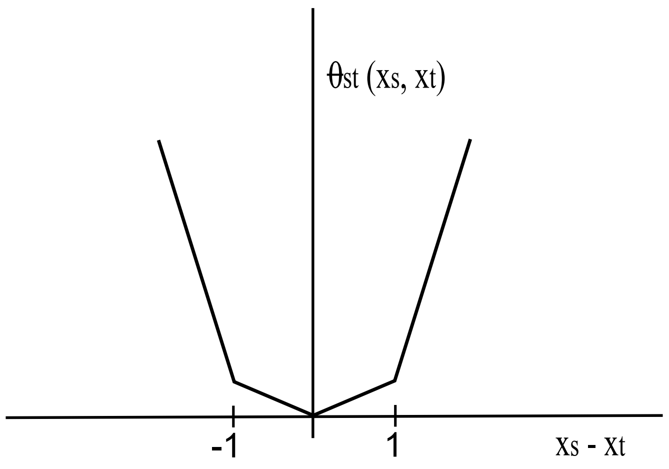

Figure 3.8: Penalization function derived from discontinuity.

Energy of configuration $\mathbf{x}$ is defined by:

$$\mathrm{E}(\mathbf{x}|\theta) = \textstyle\sum_{s\in\mathcal{V}} \theta_s(x_s) + \sum_{st\in\mathcal{E}} \theta_{st}(x_s, x_t),$$

where $\theta_s(x_s)$ is data term and $\theta_{st}(x_s, x_t)$ is a pairwise interaction term.

Graph vertices represent pixels (or blocks of pixels) in target image and their labeling value defined translation of pixel position in the mapping to the reference image. Graph has two layers of vertices, which represents via their labeling x and y displacements of pixels $\mathcal{V} = \mathcal{V}^1 \cup \mathcal{V}^2$, where $\mathcal{V}^1 \sim \mathcal{V}^2$ ($\mathcal{V}^1, \mathcal{V}^2$ are isomorphic). Edges are sets of intralayer and interlayer edges $\mathcal{E} = \mathcal{E}^1 \cup \mathcal{E}^2 \cup \mathcal{E}^{12}$, where $\mathcal{E}^1$ $(resp.\ \mathcal{E}^2) = \{(s,t) \mid s,t \in \mathcal{V}^1\ (resp.\ \mathcal{V}^2)\}$ and $\mathcal{E}^{12} = \{(s^1, s^2) \mid s^1 \in \mathcal{V}^1, s^2 \in \mathcal{V}^2, s^1 \sim s^2\}$. Intralayer edges are composed of a set of horizontally and vertically neighboring pixel pairs in both layers. Interlayer edges lead from a vertex in the first layer to the corresponding vertex in the second layer, where this correspondence is an isomorphism.

The algorithm is based on searching configuration that minimizes the energy $min_{\mathbf{x}}\ E(\mathbf{x}|\theta)$. Intralayer edge potentials ensure continuity of obtained mapping. They can be defined as:

$$\theta_{st}(x_s, x_t) = max(c_1(|x_s - x_t| - 1), c_2|x_s - x_t|), \quad \text{for } st\ \in \mathcal{E}^1 \cup \mathcal{E}^2$$

where c_1, c_2 are constants, $c_1 \gg c_2 > 0$, which penalize transformation discontinuity higher than one pixel dramatically more than lower discontinuity (see Fig. 3.9). This causes that close pixels are forced to be transformed to the close locations. The low penalization for weak discontinuity enables better deformation field adaptation also for the other affine transformations in addition to translation, so that it is more suitable than initial expression $\theta_{st}(x_s, x_t) = abs(x_s - x_t)$.

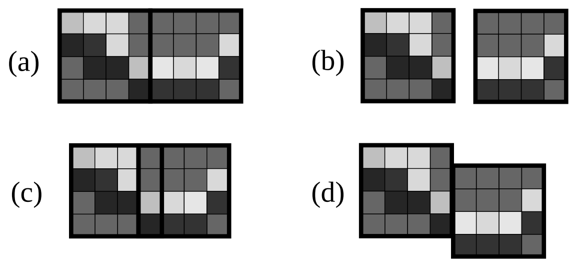

Figure 3.9: (a) Two neighboring blocks of 4×4 pixels; (b)-(d) Examples of block displacements which would be only "soft" penalized. It enables better representation of the image transformation, e.g. scaling [14].

Interlayer edge potentials ensure data similarity for minimized mapping via correlation of pair of image fragments. They can be defined as:

$$\theta_{st}(x_s, x_t) = (I_s^1 - I_{s+(x_s,x_t)}^2)^2/2\sigma^2, \quad \text{for } st \in \mathcal{E}^{12},$$

where I_s^1 and I_s^2 are fragments of reference and target image and σ^2 is expected noise variance of images.

The probability distribution $p(\mathbf{x}|\theta) = exp(-E(\mathbf{x}|\theta))$ is Gibbs distribution and certain Markov random field is characterized by this distribution. Problem of finding maximum of this Markov random field corresponds to problem of minimizing energy $\mathrm{E}(\mathbf{x}|\theta)$.

The division of finding energy minimum in x and y pixel shifts via divided layers decreases number of kept variational variables from $\mathcal{O}(|\mathcal{V}||\mathcal{L}|^2)$ to $\mathcal{O}(|\mathcal{V}||\mathcal{L}|)$.

The exact deformation model definition can be found in [14].

3.4.2 Algorithm design

The problem of finding configuration with minimal energy is in general NP-hard problem, so the approximate minimization algorithms are being developed (e.g. graph cuts, max-product belief propagation, max-product tree-reweighted message passing). However, these techniques can be applied to tree-structured graph only, otherwise convergence is not guaranteed. We have to create an edge-disjoint cover of graph $\mathcal{G}$, $\mathbf{T} = \{(\mathcal{V}^i, \mathcal{E}^i) \mid i \in I\}$, where $\mathcal{E}^i \cap \mathcal{E}^j = \emptyset, i, j \in I; \bigcup_{i \in I} T^i = \mathcal{G}$. This cover can be easily composed of three tree-structured subgraphs: horizontal chains from both graph layers, vertical chains from both layers and graph with one-edge chains corresponding to interlayer edges. In this decomposition every vertex is included in three subgraphs, in one tree from each of these subgraphs, i.e. a row, a column and an one-edged tree between layers. Minimization can be then performed by minimization in each subgraph and subsequent averaging.

Let $\Omega_{\mathcal{G},\mathcal{L}} = \mathbb{R}^{(\mathcal{V}\times\mathcal{L})\cup(\mathcal{E}\times\mathcal{L}^2)}$ be a state space of parameter vector θ, which is a concatenated vector of $\theta_s(x_s)$ and $\theta_{st}(x_s, x_t)$, $\theta \in \Omega_{\mathcal{G},\mathcal{L}}$. Let $\mathrm{E}(\mathbf{x}|\theta)$ be expressed as scalar product $\langle\theta, \mu(x)\rangle$, where $[\mu(\mathbf{x})]_s(x) = \delta_{\{x_s=x\}}, s \in \mathcal{V}, x \in \mathcal{L}; [\mu(\mathbf{x})]_{st}(x, x') =$

$\delta_{\{x_s=x\}}\delta_{\{x_t=x'\}}, st \in \mathcal{E}, x, x' \in \mathcal{L}$, where $\delta_{\{A\}} = 1$ if A is true and 0 if A is false. Let $\boldsymbol{\theta}$ be a collection of parameter vectors $\boldsymbol{\theta} = \{\theta^i \in \Omega_{\mathcal{G},\mathcal{L}} \mid i \in I\}$. For $\boldsymbol{\theta} = \sum_i \theta^i$, the value $\mathrm{LB}(\boldsymbol{\theta}) = \sum_i \min_x \langle \theta^i, \mu(x) \rangle$ is lower bound on the optimal energy:

$$\mathrm{LB}(\boldsymbol{\theta}) \leq \min_x \langle \theta, \mu(x) \rangle.$$

As we decomposed graph $\mathcal{G}$ to the tree-structured subgraphs, minimization problems are tractable for each subgraph and therefore computation of $\mathrm{LB}(\boldsymbol{\theta})$ is tractable.

Minimization problem for tree-structured graph can be solved using dynamic programming. Let T^i be a chain (each our subgraph is composed from chains). Let Φ^i_s be a min-marginal of vertex s $\in \mathcal{V}$ in tree i, which can be computed as $\Phi^i_s = \overrightarrow{\Psi}^i_s + \theta^i_s + \overleftarrow{\Psi}^i_s$. Let Ψ^i_s be incomplete min-marginals, $\Psi^i_s = \overrightarrow{\Psi}^i_s + \overleftarrow{\Psi}^i_s$, where $\overrightarrow{\Psi}^i_s(x_s) = \min_{x_t}\{\overrightarrow{\Psi}^i_t(x_t) + \theta^i_t(x_t) + \theta^i_{st}(x_s, x_t)\}$ if t is the vertex previous to vertex s in the chain T^i and $\overrightarrow{\Psi}^i_s(x_s) = 0$ if s is the first vertex in the chain. Analogously for $\overleftarrow{\Psi}^i_s(x_s) = \min_{x_t}\{\theta^i_t(x_t) + \theta^i_{st}(x_s, x_t) + \overleftarrow{\Psi}^i_t(x_t)\}$ if t is the vertex next to s in the chain T^i and $\overleftarrow{\Psi}^i_s(x_s) = 0$ if s is the last vertex of chain. By means of min-marginals we are able to find minimal labeling for each vertex in the chain. Now we can formulate an algorithm for finding optimal lower bound configuration.

3.4.3 TRW-S

We have implemented sequential TRW algorithm as suggested in [10]. The algorithm iteratively updates θ_s and thereby it increases lower bound. It performs the minimizing and the averaging step sequentially for each vertex of both layers.

Algorithm 1 TRW-S

```
1. Initialize $\theta^i_s = \frac{1}{|I_s|}\theta_s$, $i \in I$, $I_s = \{i, s \in T^i\}$.
2. For each $s \in \mathcal{V}$ sequentially perform:
    2.1 Compute $\Psi^i_s$, $i \in I_s$ on the tree $T_i$ using dynamic programming
    2.2 Update $\theta^i_s = \frac{1}{|I_s|}$ ( $\theta_s + \sum_{j \in I_s} \Psi^j_s$ ) - $\Psi^i_s$, $i \in I_s$
3. Compute actual lower bound $\mathrm{LB}(\theta^k) = \sum_i min_x \langle \theta^i, \phi(x) \rangle$
4. Check the stopping criterion, if not satisfied go to step 2.
```

Incomplete min-marginals for each vertex are kept in memory which enables faster further computation. Updates for intralayer edges can be easily computed due to divided layers $\mathcal{V}^1, \mathcal{V}^2$ for x and y shifts with complexity $\mathcal{O}(\mathcal{L}^2)$. However, there exists a method which computes the update in $\mathcal{O}(\mathcal{L})$ [5]. Updates for interlayer edges are slower and it is necessary to prepare block correlations from reference and target image before starting the computation.

Stopping criterion can be based on convergence of lower bound. Algorithm creates a sequence $\{\boldsymbol{\theta}^k\}$ performing sequential maximization of $\mathrm{LB}(\boldsymbol{\theta})$. There are fixed points in the algorithm possessing the property, called Weak Tree Agreement(WTA), each x_s from minimizing configuration $\mathbf{x}$ is equal in all chains, so we can choose local minimizers x_s from arbitrary tree as the resulting labeling, which determines most suitable block displacement.

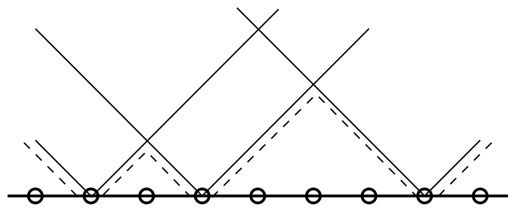

Figure 3.10: A grid with $N = 9$ points containing the point set $P = \{1, 3, 7\}$. The cones are v-shaped of slope 1. The distance transform as height of lower envelope, depicted as a dashed contour. The example is reprint from [5].

3.4.4 Method of finding incomplete min-marginals in $\mathcal{O}(\mathcal{L})$

As the computation of incomplete min-marginals is one of the key step in the iterative deform matching algorithm, it has to be done very efficiently. $\overrightarrow{\Psi}^i_s$ and $\overleftarrow{\Psi}^i_s$ are enumerated for every algorithm iteration $4 \cdot |\mathcal{V}|^2$ times (for horizontal and vertical chains left and right min-marginal). The computation of $\overrightarrow{\Psi}^i_s(x_s)$ shows the following formula,

$$\overrightarrow{\Psi}^i_s(x_s) = \min_{x_t}\{\overrightarrow{\Psi}^i_t(x_t) + \theta^i_t(x_t) + \theta^i_{st}(x_s, x_t)\},$$

where t is the vertex previous to s in the chain. The formula has to be evaluated for every $x_s \in \mathcal{L}$. The elements $\overrightarrow{\Psi}^i_t(x_t)$ and $\theta^i_t(x_t)$ are computed from previous processing and $\theta^i_{st}(x_s, x_t)$ is defined as:

$$\theta^i_{st}(x_s, x_t) = max(c_1(|x_s - x_t| - 1), c_2|x_s - x_t|).$$

Therefore, $\overrightarrow{\Psi}^i_s(x_s)$ can be easily computed in $\mathcal{O}(\mathcal{L}^2)$ $(x_s \in \mathcal{L},\ x_t \in \mathcal{L})$. However, as $\theta^i_{st}(x_s, x_t)$ is not dependent on the exact values x_s and x_t, but only on their absolute difference, a method based on Viterbi algorithm finds $\mathcal{L}$-times minimum in $\mathcal{O}(\mathcal{L})$.

This method can be easily explained by means of the distance transform problem for a 1D grid with N locations containing the point set P in this grid. The distance transform of P specifies for each grid location the distance to the closest point in the set P. The distance transform can be expressed as:

$$D(j) = \min_{i \in P}(\rho(i - j) + \delta(i)),$$

where $\rho(x)$ is a distance function, e.g. $\rho(x) = |x|$, and $\delta(i)$ is an indicator function for the set P such that $\delta(i) = 0$ for $i \in P$ and $\delta(i) = \infty$ otherwise. Distance function creates an "upward facing cone", which is rooted at each grid location with point from P. On the picture 3.10 we can see an example of a grid containing the point set P, using distance function $\rho(x) = |x|$. The distance transform at each point is given by the height of the lower envelope of the cones.

We can correctly compute this distance transform using a simple two-pass algorithm. First of all, the vector $D(j)$ is initialized to $\delta(j)$. Then two passes are necessary for finding minimal distance. In the forward pass, each successive element $D(j)$ is set to the minimum of its own value and the value of previous element incremented by one.

$$D(j) = min(D(j), D(j-1)+1), \text{ for } j = 1 \text{ ... } (n-1)$$

The backward pass is analogous:

$$D(j) = min(D(j), D(j+1)+1), \text{ for } j = (n-2) \text{ ... } 0.$$

Now we can adapt this algorithm to our problem. Our grid has $|\mathcal{L}|$ locations representing x_s values and our "cones" are placed in each grid location. They represent $\theta^i_{st}(x_s, x_t)$, so that their shape is defined by penalization function. They are rooted in $\overrightarrow{\Psi}^i_t(x_s) + \theta^i_t(x_s)$ instead of 0. The lower envelope of these cones is our searched $\overrightarrow{\Psi}^i_s(x_s)$, see Fig. 3.11.

The initialization of $\overrightarrow{\Psi}^i_s(x_s)$ by value $\overrightarrow{\Psi}^i_t(x_s) + \theta^i_t(x_s)$ is valid, because if $x_s = x_t$ then $\theta^i_{st}(x_s, x_t)$ is zero. Let $X(x_s)$ be a variable array representing actual x_t which minimizes $\overrightarrow{\Psi}^i_s(x_s)$. We can initialize $X(x_s) = x_s$, for each $x_s \in \mathcal{L}$. The forward and the backward pass can be then performed as in the algorithm below:

Algorithm 2 Finding minimum for each x_s in two passes

```
for all $x_s \in \{1, ..., |\mathcal{L}|-1\}$ do
  $D(x_s) = min(D(x_s), D(x_s-1) + \theta^i_{st}(x_s, X(x_s-1))$
  if $D(x_s)$ was changed then
    $X(x_s) = X(x_s-1)$
  end if
end for
for all $x_s \in \{|\mathcal{L}|-2, ..., 0\}$ do
  $D(x_s) = min(D(x_s), D(x_s+1) + \theta^i_{st}(x_s, X(x_s+1))$
  if $D(x_s)$ was changed then
    $X(x_s) = X(x_s+1)$
  end if
end for
```

After this two passes minimums for each x_s are found. It needs $2 \cdot |\mathcal{L}|$ comparisons, so that computational complexity of this algorithm is $\mathcal{O}(|\mathcal{L}|)$.

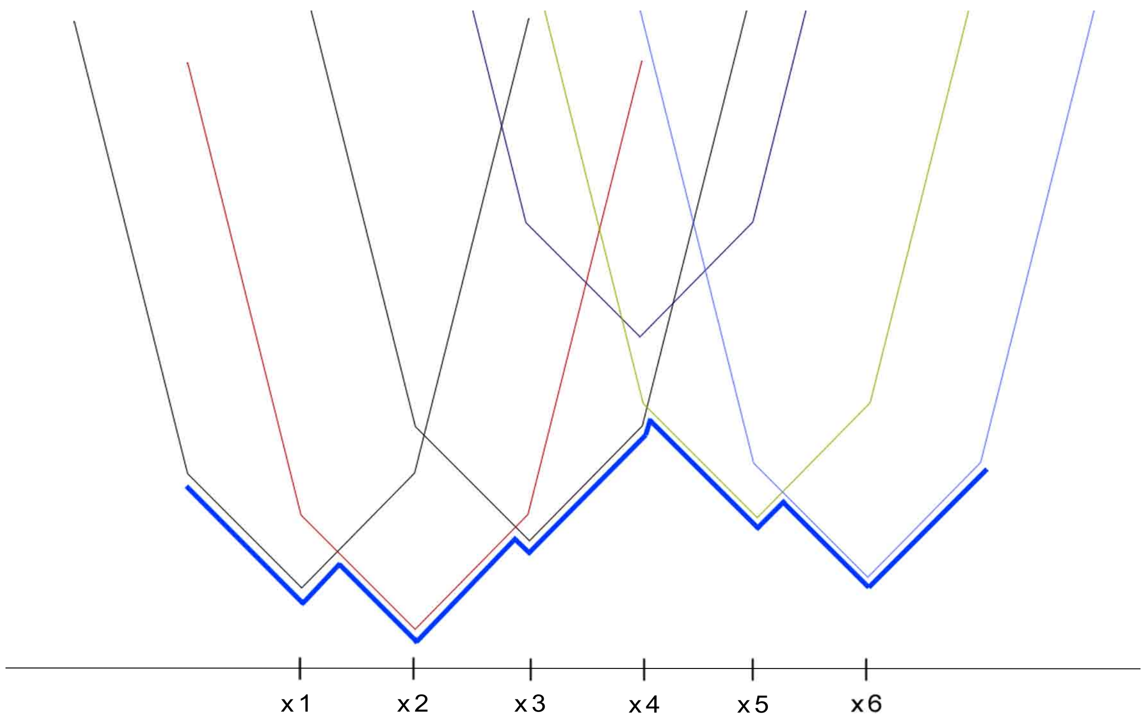

Figure 3.11: One-color functions $\theta^i_{st}(x_s, x_t)$ shifted vertically by the value $\Psi^i_t(x_s) + \theta^i_t(x_s)$, where x_s is $x_1 ... x_6$. The minimum of all these functions at each location is a lower envelope, depicted as bold contour.

Chapter 4

Realization

4.1 Platform choice

Firstly, we have to choose a platform and a programming language for our project. It is suitable to choose C/C++ language with respect to the high computation requirements of the project. The project uses STL libraries for conforming implementation and better abstraction via objects. We have used the library WxWidgets for image manipulation and OpenCV for operations with images. Another library used in project for database manipulation is MySQL connector enabling access to MySQL database from C/C++ code.

The project has been primarily developed for MS Windows, but C/C++ environment enables easy portability to other platforms (e.g. Linux). All used libraries WxWidgets, OpenCV and MySQL are also easy portable.

4.2 Project structure

Project consists of two parts, which differs in process of texture acquisition and matching process used for warping texture to reference one. Both methods consist of three parts of extracting and warping textures and building final textures, see Fig. 4.1.

The first method uses information about texture borders, collected in the application LangweilGUI, which was specially developed for this purpose. This information is loaded from database and with respect to these borders, textures are extracted from each camera. The best texture among them is chosen and features (SIFT descriptors) are detected in each image. Texture warping is performed via correspondences found by means of these features. The final texture is composed from these warped textures.

Second method uses textures acquisition by means of program TextureBuilder[9], which are extracted via reprojection of the wired 3D model. These extracted textures are warped using a deformation model. Final textures are built up from warped textures with respect to their local sharpness, occlusion and observation angle.

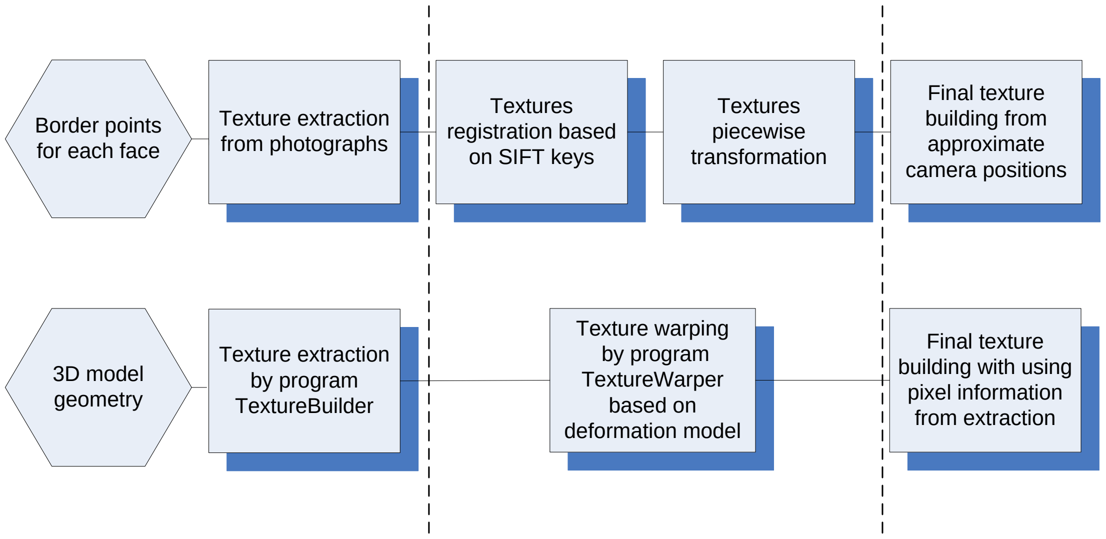
Border points for each face
Texture extraction from photographs
Textures registration based on SIFT keys
Textures piecewise transformation
Final texture building from approximate camera positions
3D model geometry
Texture extraction by program TextureBuilder
Texture warping by program TextureWarper based on deformation model
Final texture building with using pixel information from extraction

Figure 4.1: Workflow of texture processing with two different data inputs.

4.3 Feature method

We have chosen to implement the feature method based on a SIFT keys because of their robustness against changes of illumination, scale changes, range of affine distortion, but also against changes in 3D viewpoint for non-planar surfaces.

4.3.1 Texture acquisition

This method uses information about border point positions on textures and their correspondences through different images. The process of texture acquisition with known border points is quite easy. Information about border points corresponding to one face are found in database and textures are simply cut out of the images via circumscribed rectangle and masked according to their borders. Except the position of border point, information about its visibility is saved too. This can help registration algorithm by supposing that another object occludes the texture in this point neighborhood, so that feature correspondence from this area are not very trustful. There is one more information about occlusion. When any item protrudes above the face, so that there is actually a hole in the face geometry, it is also marked and its approximate position is saved in the database.

After creating each texture corresponding to one face, we are able to compute their sharpness by image gradients and reduce textures which are significantly more blurred than others.

4.3.2 Warping textures

In warping process we choose the extracted texture with the largest area that is sharp enough to be the correspondence texture, all other textures are target textures.

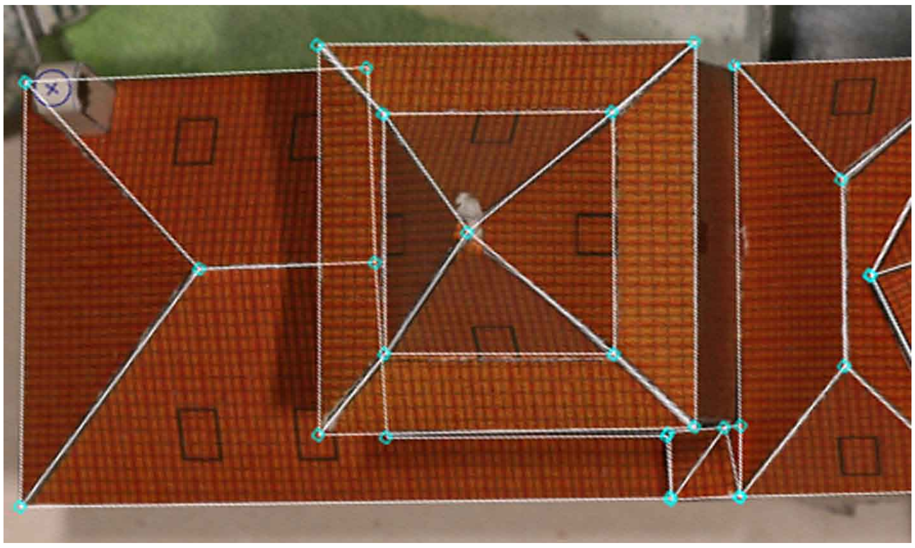

Figure 4.2: Example of the textures with marked border points and edges [1]. In the top left corner the chimney is also marked as a salient object for a higher quality of warping process.

Correspondences are found by means of SIFT descriptors. For finding keypoints and computing their descriptors we used a well-known implementation from Sebastian Nowozin [3], which generates keypoints for an image file. Feature matching is realized via Euclidean distance of keypoint descriptors from different images. For each reference image key the nearest key from target image is selected. If distance of the second nearest key is at least 80% of the nearest key distance, correspondence is not used. For fast searching the nearest key, a tree is build including all the descriptor values. Branches leading from the root of the tree represent all first descriptors values. A tree is then branching only for position which the resting keys differ at. The minimal difference is then depth-first searched. As difference minimum is decreasing more and more branches don't have to be searched through, because their partial difference is bigger than the already found difference. Correspondences obtained by this way are then further filtered.

Firstly, we can filter apparent mismatches which differ significantly from most matches. Providing that we approximately know border correspondence of textures, matches can be filtered also according to this information. For this purpose homography matrix defined by corresponding border points is used. Even for significantly curved textures this filtering can reject some useless mismatches. Furthermore, we used a grid for detection of the locally not trustful correspondences. Within this testing, a correspondence is compared to the corresponding points falling to the neighboring cells. A harder threshold can be used for this local correspondence filtering, because even if our texture faces are not exactly planar, they are continuous and we can assume a continuous change in correspondence orientation. Finally, filtering is performed also for decreasing density of keypoints in places where their number is sufficient and where it would only increase computational complexity of all following steps without appropriate effect on the final quality of the texture. This reducing is done using a grid. When more correspondences fall into one cell, only

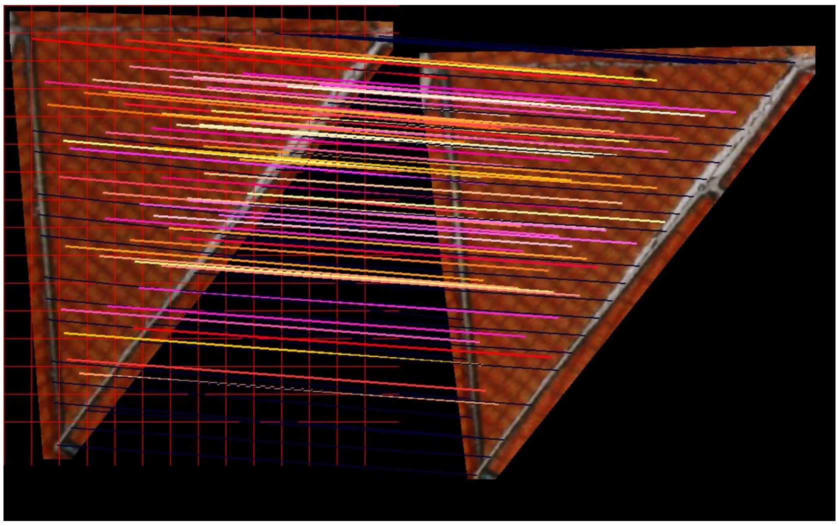

Figure 4.3: Color lines representing SIFT correspondences in two different images filtered in area by a grid.

the more trustful ones rest (see Fig. 4.3). The trust in correspondence is given by similarity of their descriptors and by the ratio of the nearest point descriptor and the second nearest one.

In case that SIFT keys are not enough dense in the image, which occurs e.g. for small or narrow faces or faces with almost constant color texture, their image size can be doubled before processing, which leads to extraction of substantially more keypoints. It may occur that we don't have enough correspondence anyway, but it is usually in cases that it doesn't matter e.g. almost color constant textures don't have to be warped very precisely.

On the other hand, when we have enough trustful correspondences in the location of given border point, we are even able to place border points in the target image more precisely. Their improvement is possible in sense of their location in reference image. If the border point is marked wrongly in the reference image, border point in the target image can be moved to this wrong location. The improvement of wrong border point locations can't be usually easily estimated (e.g. by edge intersection detector) because of the texture structure including elements, which can be wrongly thought of as border points, such as tiles on the roof, or window corners on the wall.

After correspondence filtering, the target image is transformed to the form of reference image via found correspondences. It is done piecewise using a triangulation created from corresponding points. The triangulation is built up only in the reference image and points on the border lines are added for improvement of the triangle set. Then each triangle from the target image is transformed to the triangle in the reference image. For each pixel in the reference image, the corresponding pixel in the target image is computed using the barycentric coordinates. In spite of the fact that the barycentric coordinates conserving the length ratio aren't very suitable for perspective projection, they are used here on small areas only, so that it doesn't cause any perceivable errors. Moreover, we were assuming only the knowledge of

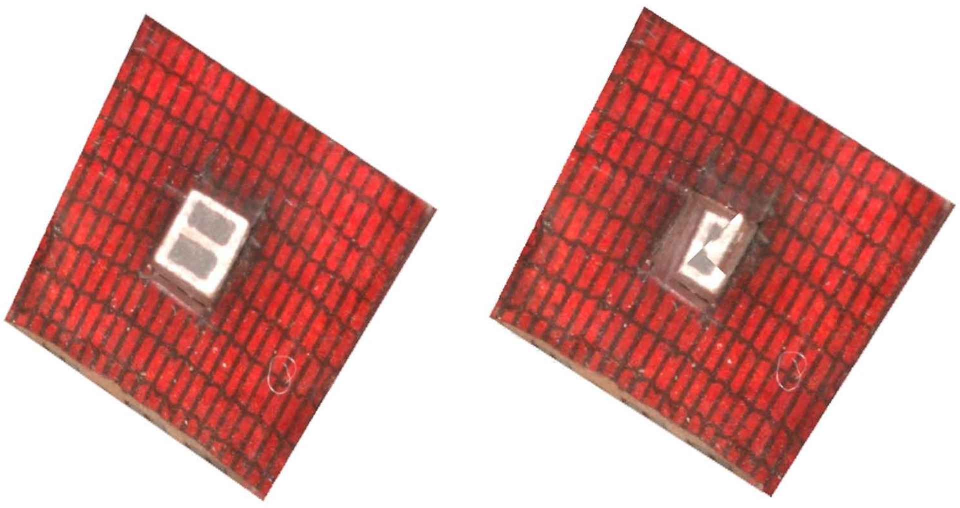

Figure 4.4: Texture built from different views around chimney [1].

texture border points in 2D at the beginning of this part, which doesn't get us depth information. After computing values corresponding to the reference image pixel, we get coordinates in target image using nearest-neighbor interpolation.

4.3.3 Building final textures

The fact that we don't know anything about texture occlusion by other textures makes the building difficult. We can assume that the object occluding our texture will change its position in views from different images. If it doesn't, it probably occludes part of the texture from all views and this texture part doesn't have to be created, because it is not visible from any views. Then places where a few textures differ from the others are possibly occluded and they are rather not used for building the final texture.

If there is an occluding object near the polygon for which we're generating texture, we need to build the texture from images from cameras with varied position and orientation. We can use approximate information about the external camera calibration to choose which images to use to assemble each part of the resulting texture.

4.4 Deformation model method

This method is based on registration by means of deformation model. Texture acquisition process was performed via program Extractor developed by Jan Kirschner[9]. The program loads 3D model from .rzi file and extracts textures using reprojection from all cameras. It prepares also masks with local occlusion and sharpness. Textures can be acquired also by other programs, which extract textures from a model.

4.4.1 Warping textures

For warping process is chosen one texture as a reference texture according to its total visibility area. Other textures are warped to this one. Number of warped textures can be reduced for speeding up the warping process according to their size and local sharpness.

We have implemented the algorithm as suggested in the previous chapter. First of all, algorithm constants are set. Block size is adjusted to image size, a smaller image has a smaller block size for better detail representation and speeding up computation for huge images. Size of set $\mathcal{L}$ is adaptively set according to the difference in the reference and the target image size. Maximal $|\mathcal{L}|$ value can be set in configuration file, default x and y allowed displacement is $\mathcal{L} = \{-30, ..., 30\}$. The higher value $|\mathcal{L}|$ is suitable for textures created for faces which have high real curvature and therefore their textures differ a lot on images taken from different views. The size of the set $\mathcal{L}$ expresses the ability of deformation field to find corresponding points which differ in x and y pixel position maximally by $|\mathcal{L}|/2$.

The second step is value precomputing. It is necessary to prepare block correlations for each pair of reference and target image before start with deformation. These correlations are computed from bright-balanced images for getting satisfying results. Block correlation precomputing is quite time-consuming step of the algorithm, but it enables quick deformation computing. Memory requirements are significant, especially for large textures. Our solution enables using image masks, which decreases time and memory requirements during the correlation computing as well as within algorithm iterations.

After precomputing correlations, graph representation structure is allocated and initialized. We set the initial values θ_s^i to 0. Min-marginals are computed at the first iteration and their values are recomputed only when any value in the chain has changed. For intralayer updates, corresponding to the continuity term, we used $\mathcal{O}(\mathcal{L})$ method based on Viterbi algorithm. For interlayer updates, corresponding to correlation term, we used $\mathcal{O}(\mathcal{L})^2$ method, however, number of these updates is lower. The stopping condition is based on convergence of the lower bound of $\boldsymbol{\theta}$ and also on the ratio of lower bound increments and time of one iteration.

Memory requirements for block correlation are $N_b \cdot |\mathcal{L}|^2$, where N_b is the number of blocks in the image, which can be computed from the image size and the block size as: $N_b = hres \cdot vres/(b_{sz})^2$. Memory used for graph representation is given by number of incomplete intralayer min-marginals $N_b \cdot |\mathcal{L}| \cdot I'_s \cdot 2$, where I'_s is the number of intralayer trees each vertex is part of. For our solution $I'_s = 2$, where number 2 represents left and right part of the chain $(\overrightarrow{\Psi}{}_s^i, \overleftarrow{\Psi}{}_s^i)$ plus number of incomplete interlayer min-marginals $N_b \cdot |\mathcal{L}|$ (as only one intralayer chain leads from vertex) plus number of actual values of θ_s^i, which corresponds to the number of intralayer min-marginals. Therefore total amount of variables is $(4 + 1 + 4) \cdot N_b \cdot |\mathcal{L}| = 9(N_b \cdot |\mathcal{L}|)$. As $|\mathcal{L}|$ is usually a few times greater than 9, block correlations represent greater memory requirements than a graph.

Transformation defined by the resulting deformation field doesn't have to be always continuous, especially at the texture borders when the reference image differs from the target one in location of unmasked area. Therefore we use a median filter on the graph before we start with the target image transformation. As our block model is able to find deformations including scaling, the neighboring blocks in the target image can be transformed to the blocks with overlapping parts. Alternatively, neighboring blocks in the target image can be transformed to the not directly neighboring blocks in the reference image. These overlaps can be simply solved by averaging color values of these double-filled pixels. The holes are filled by interpolation of the neighboring values weighted by their distance from the overlapped pixel. The nearest neighbor algorithm is not a good approach in this case because the holes in the deformation can be also wider than a few pixels and unpleasant boundaries could be created.

4.4.2 Building final textures

After we have warped all textures to the reference one, we can start with the process of building the final texture. For composing we use information gathered by extracting about sharpness, occlusion and observation angle. All these values are weighted and they create a value of pixel possible contribution. By counting contributions from each texture pixel we get total texture evaluation. These rates are used for sorting. Pixels from the highest evaluated texture are added to the final one and each pixel contribution is saved too. Pixels from other textures are then sequentially added only on so far free pixels or pixels with lower contribution value than their own. A threshold for maximal contribution value at which pixel can be overwritten by some next texture is used for preventing from senseless overwriting single pixels. This approach leads to creation of textures composed of a few source textures instead of scattered pixels from almost all sources.

4.5 Usage of developed programs

Our solution consists of two programs, Textures and TextureWarper. Program Textures is the complex program for extraction, matching, warping and building textures. It uses the program TextureWarper for warping via deformation fields.

Usage of the program Textures
Program Textures enables two different approaches to the texture extraction problem. The feature-based method of processing that works with texture border points as an input is default. The deformation model approach can be chosen by parameter *-d*. Other parameters are listed below:

- *-p Part number* Number of the model part, which should be processed. This parameter is mandatory for both approaches.

The following parameters are intended for feature-based method:

- *-f Face ID* If this parameter is used, only textures corresponding to the face with this face ID will be processed.

- *-z* Detection of the blurred textures will be performed by grouping textures according to their gradients for finding the best height for each group of images. If a value follows this parameter, the threshold is set. The photographs with z coordinate under this treshold will not be used.

The following parameter is suitable for deformation model method.

- *-r First face number*, which should be processed.

The program Textures uses the following parameters from configuration file:

- *LogFile* Filename of log file.
- *LogLevelTextures* Level of logging (0 - debugging, 1 - warning, 2 - errors).

The following parameters are intended for the feature-based method:

- *JPGPath* Path to folder containing images.
- *Texture2DPath* Path to textures extracted using border points, deformed by feature-based method.
- *DownScaleSift* Relative scale of image used for Scale-invariant feature transform.

The following parameters are suitable for the deformation model method:

- *RziFileName* Rzi file name with a model which textures were extracted according to.
- *TexturePath* Path to extracted textures prepared for warping.
- *FinalTexturePath* Path for final textures built from textures warped by program TextureWarper.
- *DeformedTexturePath* Path to textures deformed by TextureWarper.
- *MinFinalPixelWeight* Minimal weight of pixel for marking as final.
- *MaxTextureNumberVisibility* Maximal number of textures from one face that are deformed.

Usage of the program TextureWarper
Program TextureWarper takes the reference and the target image as inputs. Optionally their masks can be given as an extra file or in the alpha channel of the image. The arguments of this program should be set as:

1. The file name of the reference image.

2. The file name of the reference image mask or 0, when file name of the reference image with ending ".masks.png" instead of ".png" should be used.

3. The file name of the target image.

4. The file name of the target image mask or 0, when file name of the target image with ending ".masks.png" instead of ".png" should be used.

5. Optional argument for file name of new deformed image. If it is not used, target image will be overwritten.

The program uses the following parameters from configuration file:

- *DMLogFile* Filename of log file.
- *LogLevelDeform* Level of logging.
- *minK* Minimal used K value determining allowed block displacement $(-K, K)$ for deformation model.
- *smoothSize* Size of smoothing block for deformation field.

Chapter 5

Digitalization of Langweil's model of Prague

This thesis deals with the problem of texture creation for a 3D model. I had a unique opportunity to try out the system of texturation on real data from the project of digitalization of Langweil's model. Langweil's model of Prague is a paper and wood model that was made by Antonín Langweil and shows the centre of Prague how it looked like back in the 1830s. It depicts Old Town, Lesser Town, Prague Castle and Hradčany, see Fig. 5.2. The model is now stored in the Museum of the City of Prague[2].

5.1 Model specifications

Langweil's model is outstanding also for incredible size and details. In the area of $20m^2$ it shows more than 2,000 buildings in scale 1:480. There are many details in the model as chimneys, trees, walls that make the reconstruction difficult and require considerably high accuracy of reconstructed components. A lot of objects are also painted on the walls as windows, gates, signboards, so that the high quality textures are very desirable. As the model documents the appearance of Prague in

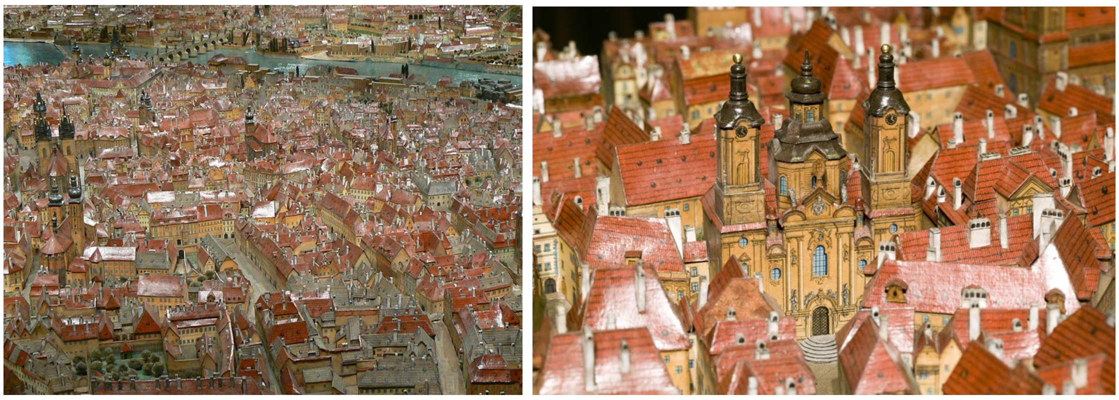

Figure 5.1: View over the model. Incredible details on the model [1].

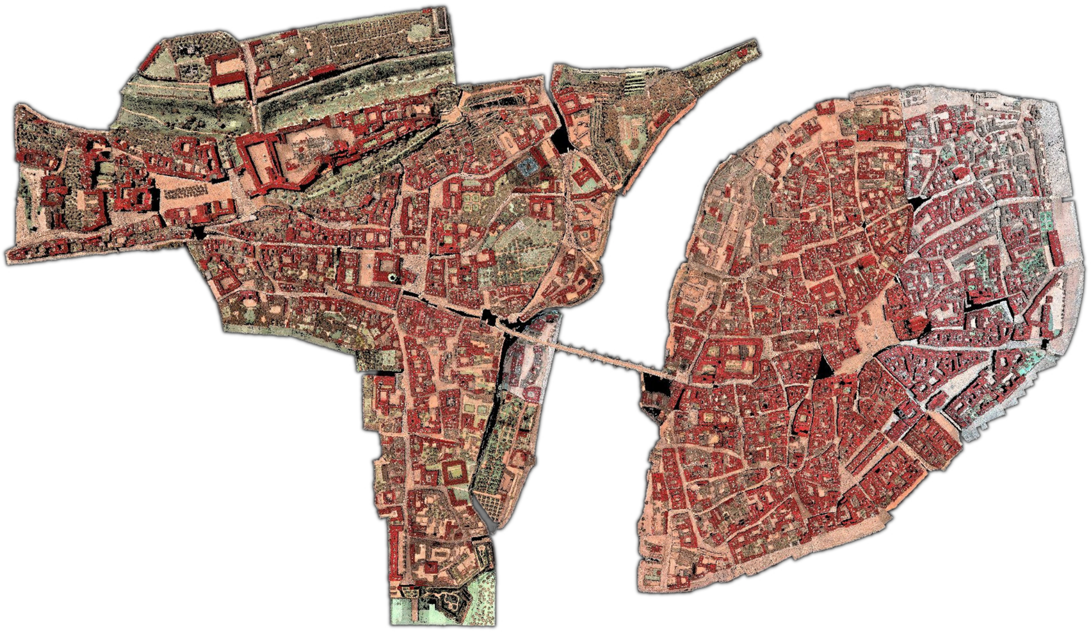

Figure 5.2: Point cloud of the entire Langweil's model [1].

the beginning of 19th century, which looks today much different, it has also a great historical value. The proper reconstruction including textures is important also for the preservation of the monument.

It would not be feasible, because of the model detail and size, to digitalize it manually, see Fig. 5.1. Traditional methods are not able to work with models of such size or they are very time-consuming. There are many obstacles arising from the fact that it is about 180 years old paper model such as high curvature of the paper, imperfection of paper joints which don't create exact edges, dust, cotton-wool left after model cleaning. These imperfections are easily observable on high quality macro photos.

5.2 Workflow

The whole process of digitalization must proceed with respect to the model size. The first stage includes taking images of the model. High color quality macro photos with high resolution were systematically captured for preparing enough data for model reconstruction with high quality textures. These pictures were 16Mpix with 13bit per each color channel and they require approximately 50MB of memory each. The pictures were taken in a grid for covering all parts with different elevation and azimuth for getting at least some pictures from hardly visible places. For dealing with the shallow depth of field a camera was taking pictures in more heights, see Fig. 5.3.

As the existing systems are not capable to work with such amount of data, whether we talk about camera calibration tools or 3D model reconstruction or tex-

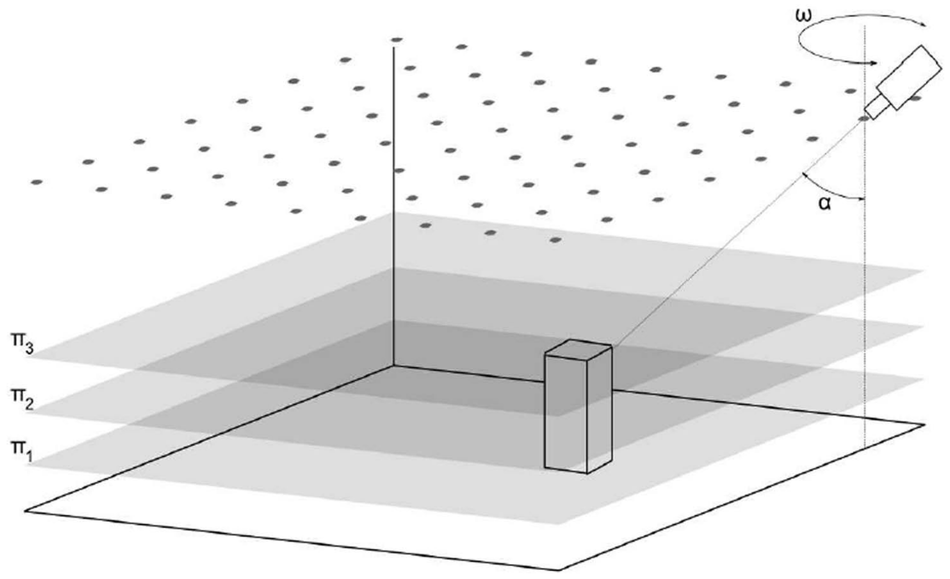

Figure 5.3: Dots on the image show systematical image capturing in a grid with 3 different camera elevations α and 8 different azimuths ω. We can see 3 different focus planes π^1, π^2, π^3 which correspond to three camera heights above the model. Courtesy of Jan Kirschner [9].

ture creation, we had to create our own workflow and own applications for solving many subtasks. Firstly we were gathering the 2D topology of model roofs from face border points and their correspondences. First textures were prepared using this collected 2D border point data. From these correspondences 3D roof geometry was created from images taken from perpendicular downward views. Then we calibrated these top images. The rest image calibrations were computed using the top image calibrations by means of a camera resectioning tool developed by Lukáš Mach [12]. A model of each object is prepared in the program ImageModeler++. Then we used the chimney detector developed by Lukáš Mach and floor detector developed by David Sedláček. Finally, by adding prepared models of trees and bushes to detected position we get the most parts of wired 3D model. For this model textures are created, see Fig. 5.4. Accuracy of the wired model vertices can be further improved according to the generated textures.

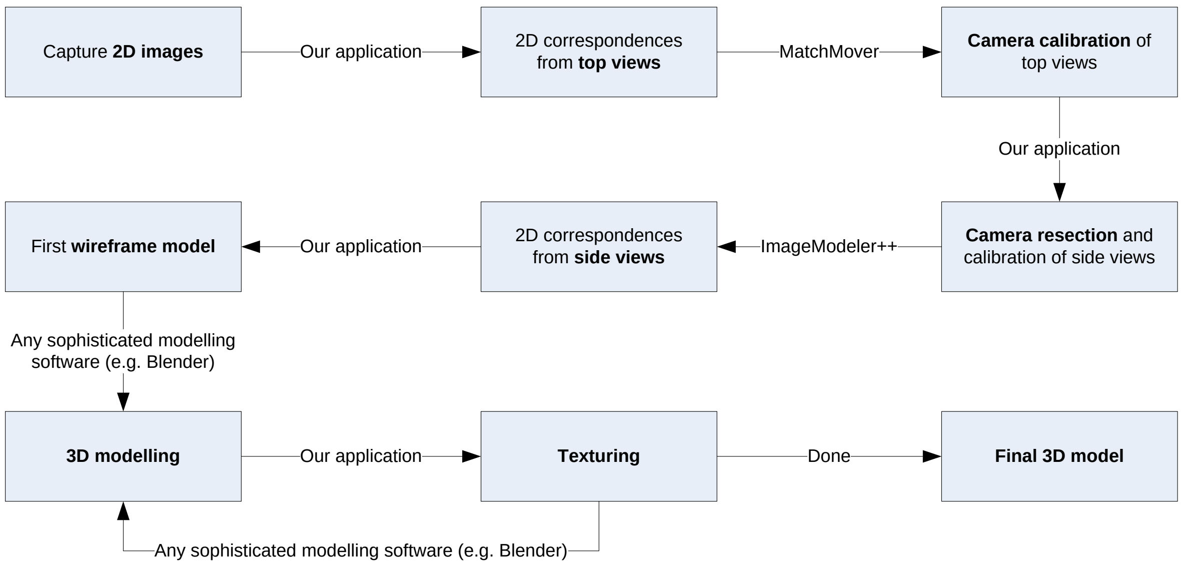
Capture 2D images
Our application
2D correspondences from top views
MatchMover
Camera calibration of top views
Our application
Camera resection and calibration of side views
ImageModeler++
2D correspondences from side views
Our application
First wireframe model
Any sophisticated modelling software (e.g. Blender)
3D modelling
Our application
Texturing
Done
Final 3D model
Any sophisticated modelling software (e.g. Blender)

Figure 5.4: Our workflow of digitalization. Courtesy of Matěj Cáha [4].

Chapter 6

Testing and method comparison

6.1 Deformation model comparison

We will compare deformation model method implemented by Shekhovtsov, Kovtun and Hlaváč [14], referred to as DeformMatch, with our deformation model, referred to as TextureWarper. Firstly, we compare transformation times for one middle-sized texture (see Fig. 6.1). The texture was deformed with using different block size Bl_{sz} and value K, which defines size of $\mathcal{L}$ as $\mathcal{L} = \{-K, ..., K\}$. Deformations by the help of the program TextureWarper were performed also with a mask of the texture, it is marked in a table by *, the ratio of unmasked part and the texture size is listed next to the concrete texture size. All measured times are averages from a few runs of program with same inputs.

Program name	Texture size	Bl_{sz}	K	Total Time
DeformMatch	328 × 327	4	30	56.9s
TextureWarper	328 × 327	4	30	14.2s
TextureWarper*	328 × 327 * 0.49	4	30	10.2s
DeformMatch	328 × 327	6	30	26.6s
TextureWarper	328 × 327	6	30	7.9s
TextureWarper*	328 × 327 * 0.49	6	30	7.7s
DeformMatch	328 × 327	4	15	32.4s
TextureWarper	328 × 327	4	15	5.6s
TextureWarper*	328 × 327 * 0.49	4	15	3.5s
DeformMatch	328 × 327	6	15	12.2s
TextureWarper	328 × 327	6	15	4.0s
TextureWarper*	328 × 327 * 0.49	6	15	2.4s

Table 6.1: Comparison of deformation models for middle-sized texture.

With regard to our tests, we can see that our program deformed the texture in shorter time. As block size determines total block count, i.e. the size of the graph representing deformation, computation is faster with larger blocks. Value K

defines maximal searched x- and y- displacements and therefore higher K means a significant slower computation and higher memory requirements. TextureWarper required approximately 54MB of memory with $K = 15$ and 116MB with $K = 30$, both for block size of 4. The warping process can be speed up using texture with a mask. This also leads to higher quality textures as surround of the texture doesn't affect computing of deformation.

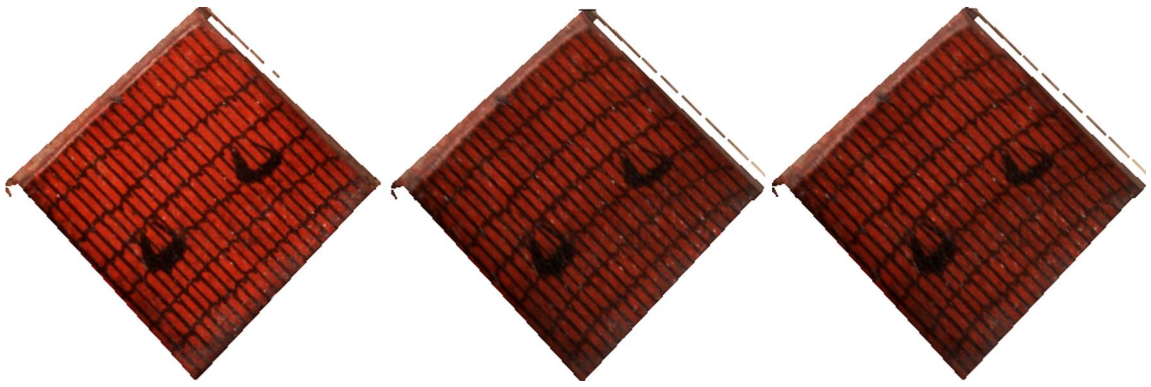
Figure 6.1: The reference image is on the left, the image deformed by program TextureWarper is in the middle and target image is on the right side [1].

The measurements were repeated with a larger texture. The reference and target texture have different focused locations, which is quite usual state. The reference image was focused at the bottom part, the target at the top of texture. The deformed textures from DeformMatch and TextureWarper are showed on Fig. 6.2.

Program name	Texture size	Bl_{sz}	K	Total Time
DeformMatch	777×271	8	15	14.5s
TextureWarper	777×271	8	15	4.1s
TextureWarper*	777×271 * 0.8	8	15	6.7s
DeformMatch	777×271	4	15	65.4s
TextureWarper	777×271	4	15	15.0s
TextureWarper*	777×271 * 0.8	4	15	15.1s

Table 6.2: Comparison of deformation models for large texture.

For the first texture we measured also time for precomputing block correlations and time needed for one algorithm iteration. Listed values were received for texture with resolution 328×327 for $K = 30$. From the Table 6.3 it is evident that the precomputation of block correlation is very time consuming part of the program. Involving the mask of the texture can even slow down this process. Time spent in one iteration is proportional to block count, so that their restriction by a mask speeds up graph iterations.

All tests were performed on computer HP xw4400 Workstation with operating system Windows XP, Service Pack 2, with 4GB RAM and CPU Intel Core 2 Duo E6700 2.66 GHz.

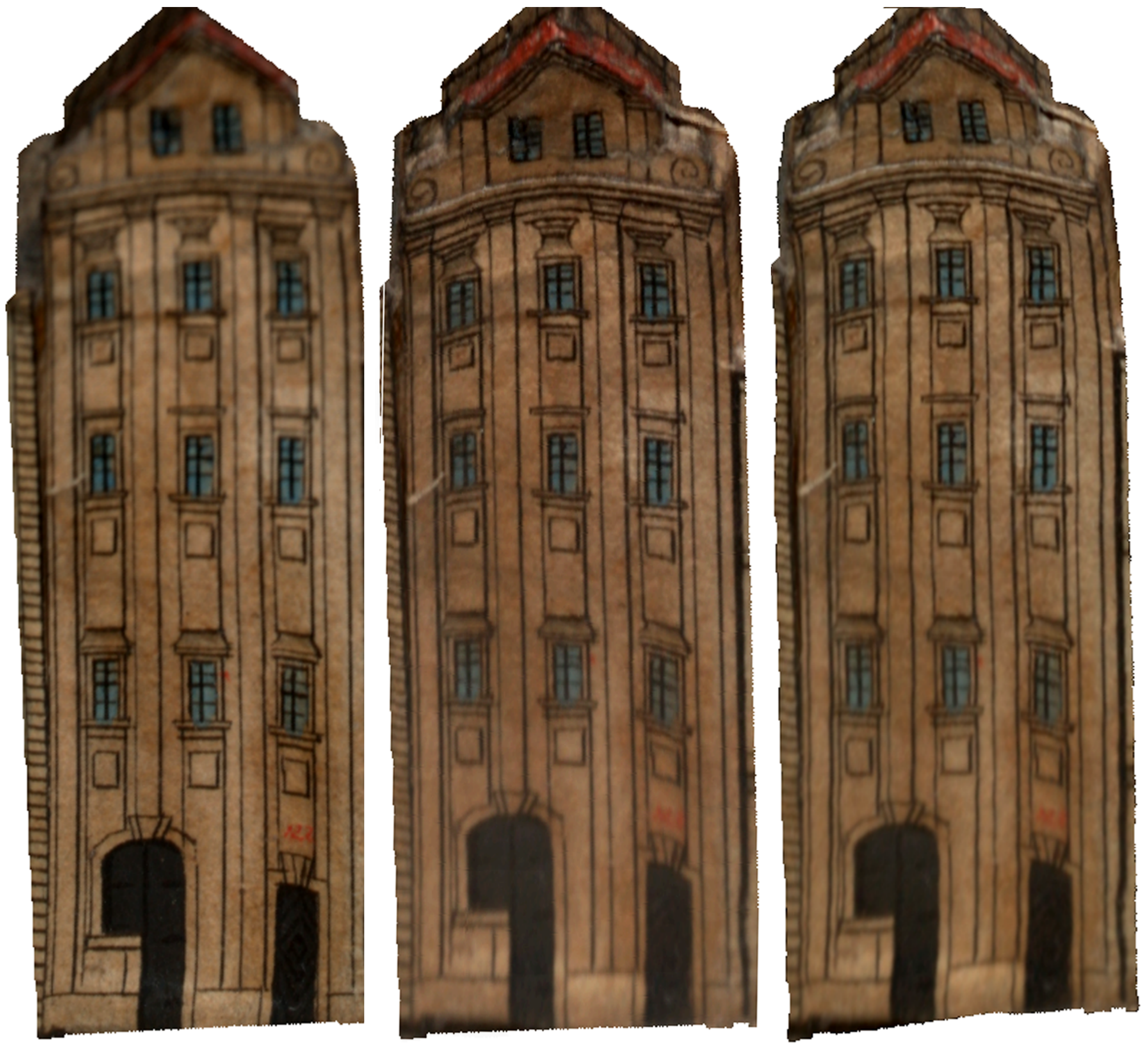

Figure 6.2: The reference image is on the left, the deformed image created by program TextureWarper is in the middle and image on the right side was deformed by program DeformMatch [1].

6.2 Feature method versus deformation model method

Our feature based and deformation model method use different input information, they search image transformation in dissimilar way, therefore also their results differ. In general the textures from deformation model method seem to be more usable due to more information being involved, especially about the overlapping faces. The insufficient knowledge about this occlusion may cause occurrence of mismatches in feature point correspondences, which cause local imperfections in texture, see Fig. 6.3.

With regard to speed of implemented method, the feature based approach is the faster one. That came from texture extraction from a model, which sequentially extracts all textures from each camera without considering their total visible size,

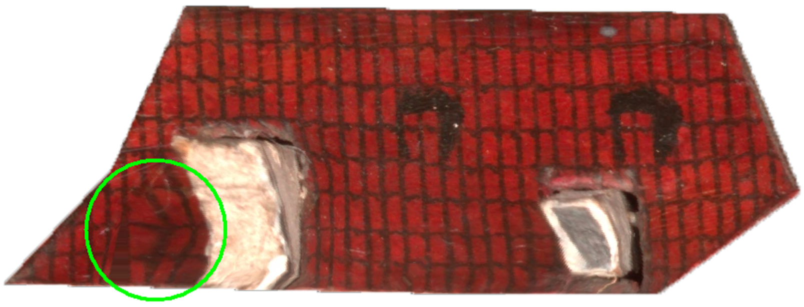

Figure 6.3: Local imperfection in texture caused by mismatching feature points [1].

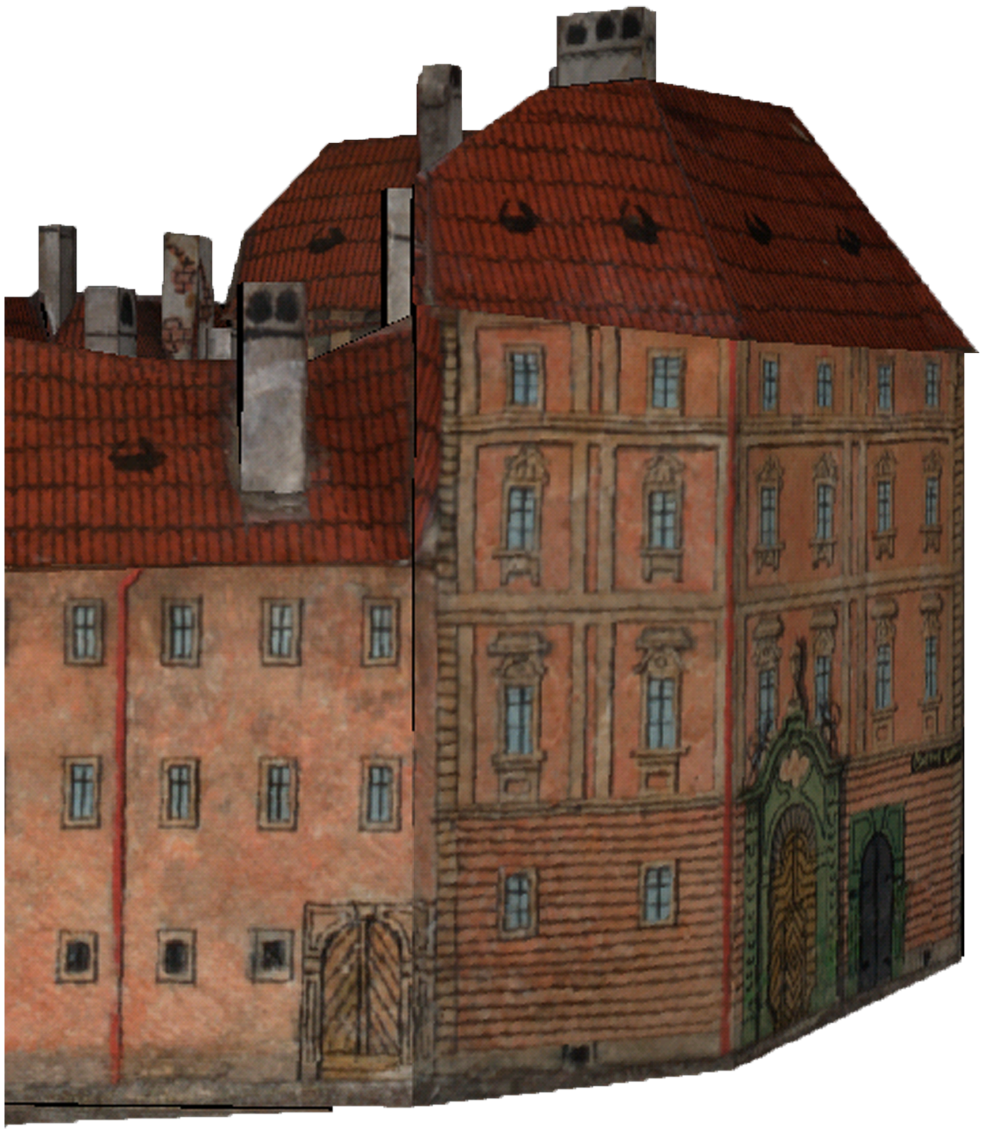

Figure 6.4: The neighboring textures don’t pass together, their structure and brightnesses differ and the texture on the right involves parts of other faces [1].

Block count	Bl_{sz}	Total Time	Correl. precomp.	Iteration time
82×81	4	14.2s	4.5s	0.547s
$82 \times 81^*$	4	10.2s	5.8s	0.265s
54×54	6	7.9s	4.2s	0.235s
$54 \times 54^*$	6	7.7s	5.7s	0.125s

Table 6.3: Comparisons of times for precomputing block correlation, total times and times for one iteration.

Method	Texture size	K	Warping Time
Feature-based method	560×500*0.5	X	39s
Deformation model method	560×500*0.5	15	37s
Deformation model method	560×500*0.5	30	105s

Table 6.4: Comparisons of warping times for 9 extracted textures processed by the implemented methods.

sharpness, etc. This leads to extraction of a large amount of textures, whose number is then restricted. On the other hand textures prepared for feature-based method are obtained from border points, which are marked on photographs where textures have large visible areas and are sharp enough, but these border points are created by hand. So that it is the question of automatization rate and speed. The warping times for both approaches are quite similar, but in the case of deformation model, it is dependent on maximal searched displacements(K). The times for same texture processed by both methods are showed in Table 6.4, where one reference and 8 target images were used.

As textures are created for each face separately, they are continuous inside the face, however problems might occur at their borders. Neighboring textures don't have to pass together in brightness, which can be balanced afterwards. Also discontinuities in the texture structure might occur at the borders of the neighboring textures as a consequence of inaccurate reconstruction, see Fig. 6.4. This problem can be solved for instance by modification of texture coordinates. It is not easy task, however, to determine their values automatically.

Chapter 7

Conclusion and future work

In this thesis, we have surveyed main methods for texture extraction and image registration suitable for model texturing. We have designed and implemented two techniques for texture creation. The feature based method which is suitable in cases when limited information about 3D model geometry is available and deformation model method which can efficiently use a given geometry for computing texture data, such as occlusions and sharpness, for their subsequent usage in the warping process. By using our solution even huge models can be textured, which could not be feasible to do manually. Both methods were tested during the digitalization of Langweil's model of Prague, which offered a wide variety of testing data.

This thesis detects some problems in the process of texture creation. For instance problem of not passing structures in neighboring textures on model could be solved by future development. Another interesting theme for future work can be the connection of benefits of both methods to a sophisticated system for texture extraction. The deformation model method could use information about most trustful correspondences obtained by the feature method. The primary tests based on adding marks to textures before warping, which represented detected features, were performed with positive results. Such method could also speed up the texturing process by allowing narrower range of deformation parameters to search through.

Bibliography

[1] DLM project photo-documentation. Courtesy of Visual Connection, a.s. and The Museum of the City of Prague.

[2] *Muzeum Hlavního města Prahy a Langweilův model Prahy.* http://hrady.dejiny.cz/praha/langweil/index.htm.

[3] *S. Nowozin. SIFT implementation - autopano.* http://user.cs.tu-berlin.de/~nowozin/autopano-sift.

[4] M. Cáha. Point correspondences registration from different image sources. Master's thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2008.

[5] P. F. Felzenszwalb, D. P. Huttenlocher, and J. M. Kleinberg. Fast Algorithms for Large-State-Space HMMs with Applications to Web Usage Analysis. *Advances in Neural Information Processing Systems*, 16, 2003.

[6] F. Fraundorfer and H. Bischof. Evaluation of local detectors on non-planar scenes. In *In Proc. 28th workshop of the Austrian Association for Pattern Recognition*, pages 125–132, 2004.

[7] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *4th ALVEY Vision Conference*, pages 147–151, 1988.

[8] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[9] J. Kirschner. *Rekonstrukce 3D modelu z fotografií.* Master's thesis, České vysoké učení technické v Praze, 2008.

[10] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *Artificial Intelligence and Statistics*, pages 182–189, 2005.

[11] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[12] L. Mach. Semi-automatic system for reconstruction of 3d scenes. Bachelor's thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2008.

[13] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *In Proc. European Conf. Computer Vision*, pages 128–142. Springer Verlag, 2002.

[14] A. Shekhovtsov, I. Kovtun, and V. Hlaváč. Efficient MRF Deformation Model for Image Matching. Research Report 8, Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Technická 2, Prague 6, Czech Republic, Aug. 2006.

[15] B. Zitová and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, 2003.

Appendix A

Configuration file example

```
[Textures]
LogFile=Textures.log
LogLevelTextures=1
RziFileName=dil_09a.rzi
DownScaleSift=1.0
TexturePath=dil_09\\textures
FinalTexturePath=dil_09\\finalTextures
DeformedTexturePath=dil_09\\deformedTextures
JPGPath=Z:\\jpg4k\\
Texture2DPath=dil_09\\textures2D
MinFinalPixelWeight=0.8
MaxTextureNumberVisibility=30

[TextureWarper]
DMLogFile=TextureWarper.log
LogLevelDeform=1
minK=30
smoothSize=10
```

```
Examples for parameter configuration of program Textures for feature-based
and deformation model methods:
  Textures.exe -p part_number [-f face_number]
    [-z [threshold_for_defocused_images]]
  Textures.exe -p part_number -d mod_deform_match
    [-r first_face_number]

Example for parameter configuration of program TextureWarper:
  TextureWarper.exe reference_image_name name_of_reference_image_mask
    target_image_name name_of_target_image_mask [name_for_new_deformed_image]
```

Appendix B

Abbreviation List

2D	Two-Dimensional
3D	Three-Dimensional
DLM	Digitalization of Langweil's model
LB	Lower bound
SIFT	Scale-Invariant Feature Transform
STL	Standard Template Library
TRW	Tree-Reweighted Algorithm
TRW-S	Sequential Tree-Reweighted Algorithm
WTA	Weak Tree Agreement

Appendix C

Contents of CD

```
|
+-src      ... source files
|     +-Textures      ... source files of the main project Textures
|     +-TextureWarper ... source files of the TextureWarper project
|     +-Textures.sln  ... project solution file for both projects
|
+-documentation
|     +-index.html    ... documentation of the source code
|
+-text
|     +-thesis.pdf    ... text of the thesis
|
+-data     ... data examples
      +-textures      ... texture samples for program testing
```