Czech Technical University in Prague

Faculty of Information Technology

Department of Theoretical Computer Science

Master's thesis

As-Rigid-As-Possible Image Registration

Marek Dvorožňák

Supervisor: Ing. Daniel Sýkora, Ph.D.

6th January 2014

Acknowledgements

I would like to thank my family and friends for support during writing this thesis. I would also like to thank my supervisor, Daniel Sýkora, for valuable advices and for a guidance throughout the process of creating this thesis.

Declaration

In Prague on 6th January 2014 . . . .

Citation of this thesis

Dvorožňák, Marek. *As-Rigid-As-Possible Image Registration.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2014.

Abstract

This thesis deals with image registration method that can be employed for image pre-processing for the purpose of facilitating the process of feature extraction. This method is particularly suitable for registering images capturing articulated objects, for example figures. The thesis includes an overview of basic deformation models employed in image registration, while it focuses on models that preserves rigidity. An overview of basic image registration methods is also included. A tool allowing to perform image registration that preserves rigidity was implemented into development version of GIMP, details of its implementation are described in the thesis. Experiments presenting its functionality were performed and their results are included in this thesis.

Keywords image registration, as-rigid-as-possible, image pre-processing, GIMP

Abstrakt

Tato práce se zabývá metodou registrace obrazu, kterou je možno použít pro předzpracování obrazu k účelu usnadnění procesu extrakce příznaků. Tato metoda je vhodná zejména pro registraci obrazů zachycujících kloubově pohyblivé objekty, např. postavy. Práce obsahuje přehled základních deformačních modelů používaných při registraci obrazu, přičemž se zaměřuje na modely zachovávající tuhost. Přehled základních metod registrace je zde také uveden. Nástroj umožňující vykonávat registraci obrazu se zachováním tuhosti byl naimplementován do vývojové verze programu GIMP, detaily jeho implementace jsou v práci popsány. Experimenty prezentující jeho funkčnost byly provedeny a jejich výsledky jsou součástí této práce.

Klíčová slova registrace obrazu, as-rigid-as-possible, předzpracování obrazu, GIMP

Contents

List of Figures

List of Tables

Introduction

The CRISP-DM standard [22] describes essential phases that are worth not forgetting during processing of data-mining project. One of these phases is *data preprocessing* phase, in which data are prepared so that they can be used in modeling phase, i.e. to build models used e.g. for prediction or classification.

The preprocessing phase is very important, an informal rule GIGO[1] is what applies here. That means, if data, that are entering the modeling phase, are redundant, are not correctly formated, contain invalid values etc., then non-functional model is often produced.

Image, in computer science often understood as a two-dimensional array, is usually the source of a large amount of redundant data. Therefore, it is often convenient to represent these data in a form other than image pixels. A face of a person can be expressed, for example, by *features* such as color of his eyes, hair or skin, eye spacing, nose size etc. Such a process of transformation of data to other, more appropriate, is called *feature extraction*.

In order to extract features from images in a simpler way, it is often useful to properly position the image, or as the case may be, to deform it before the feature extraction phase. This way, the process of extraction becomes easier, because then we can, for example, expect that person's eyes are always located at the same position. Let us look at another similar example. We analyze a video clip capturing a human figure. For each frame of the video clip and from a position of his hand, we want to extract some information (e.g. whether the hand grips some object or whether it is empty). To simplify this process, we need to know the correspondence

[1]Garbage In, Garbage Out.

between the position of the hand in each frame. Problems of this type can be solved using *image registration*.

Motivation

With methods of image registration and image deformation I familiarized in digital image processing class, where I tried to implement a method allowing to deform an image as if it were an object from the real world which is made of rubber. I was impressed by the results of image deformation using this method.

As a long-time user of free software, I long felt the need to somehow contribute to this sofware. I chose a popular free graphics program GIMP and presented my implementation of the method including comparison with several other image deformation methods to its community. I was selected to implement this method into GIMP within Google Summer of Code 2013.

Goals of the thesis

The aim of this thesis is to implement image registration method that uses the mentioned image deformation method preserving rigidity, integrate it into GIMP and compare this implementation with tools Drop and NiftyReg that allow non-linear image registration.

GIMP currently does not have a tool for image registration. This functionality can be added to it by an existing registration plugin. However, it only allows to perform affine image registration, that means registration employing affine deformation model. The tool allowing image registration preserving rigidity can therefore be interesting for GIMP.

Structure of the thesis

This thesis is divided into 4 chapters. Since deformation models are used in image registration, the first chapter includes an overview of basic deformation models and also models allowing the mentioned deformation of images that preserves rigidity.

The second chapter is devoted to image registration. Besides other things, this chapter includes some further applications of image registration, an overview of basic image registration methods and a description of image registration method that preserves rigidity.

The third chapter describes implementation of deformation algorithm and subsequent implementation of deformation and registration into GIMP. Among other things, there is also a description of user interface of developed tools.

The last chapter presents results obtained using developed image registration tool. These results are compared with the tools Drop and NiftyReg allowing non-linear image registration.

Chapter 1

Image deformation

During image registration, the source image is deformed to well align with the target image. Image deformation is therefore a key part of the process of image registration. In the same or very similar meaning as the term "deformation", terms "image warping", "image transformation" or "image distortion" are also often used.

In this chapter, we describe the basic linear deformation models as well as some interesting models allowing to perform more flexible deformation of images. An overview of image deformation methods can be found in [5] or [11].

We assume that the deformation is specified using a group of points $\mathbf{p}_i$ on the source image (source points) and corresponding points $\mathbf{q}_i$ on the target image (target points). The coordinates of the source image pixels will be referred to as (x, y), the coordinates of pixels of the deformed source image as (x', y').

1.1 Translation

The simplest image deformation method is translation. It is a shift of every source image pixel by a shift vector $\mathbf{t} = (x_0, y_0)$. Translation is thus determined by one source point and one corresponding target point. It can be expressed as

$$\begin{array}{ccccc} x' & = & x & + & x_0 \\ y' & = & y & + & y_0 \end{array} .$$

1.2 Similarity

Similarity includes a translation, a rotation by an angle α and a scaling. It is determined by two different source points and their two corresponding counterparts on the target image. It can be expressed as

$$\begin{array}{rcrcrcl} x' & = & s\cos(\alpha)x & + & s\sin(\alpha)y & + & x_0 \\ y' & = & -s\sin(\alpha)x & + & s\cos(\alpha)y & + & y_0 \end{array},$$

where s denotes a scaling factor.

When $s = 1$, this deformation is called a **rigid deformation**.

1.3 Affine model

Affine deformation further generalizes the similarity and also includes a shearing. It is determined by three different non-colinear source points and their counterparts on the target image. It can be expressed as

$$\begin{array}{rcrcrcl} x' & = & a_{11}x & + & a_{12}y & + & x_0 \\ y' & = & a_{21}x & + & a_{22}y & + & y_0 \end{array} \tag{1.1}$$

or simply as

$$\mathbf{p}' = \mathbf{Ap} + \mathbf{t}$$

where $\mathbf{p}'$ are sought coordinates of a point (pixel) on the deformed image, $\mathbf{p}$ is the coordinate of a point on the source image and $\mathbf{A}$ is an affine matrix in the form

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

1.3.1 Multipoint affine deformation

When we have exactly 3 pairs of corresponding points, we can acquire the transformation matrix $\mathbf{A}$ by solving a system of equations that can be obtained by substituting these points to equations 1.1. This way we obtain the system consisting of 6 equations of 6 unknowns.

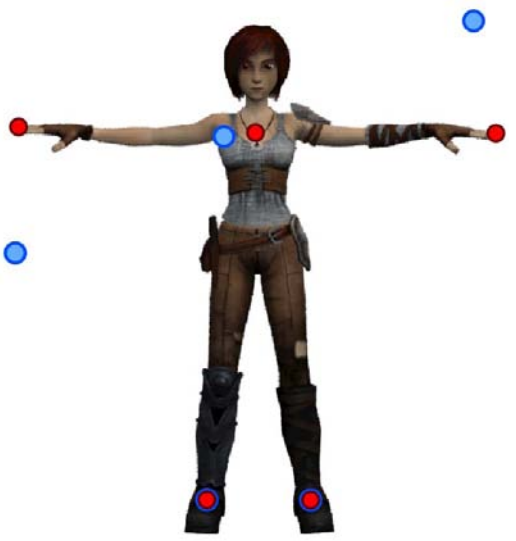


Figure 1.1: *Multipoint deformation of image*

When we have more than 3 pairs of corresponding points, we can estimate the affine deformation employing the least squares method [24, 26]. We will show how to employ this method using Figure 1.1[2].

Our task is to find such affine transformation of coordinates (i.e. transformation matrix $\mathbf{A}$ and translation vector $\mathbf{t}$) which moves red points $\mathbf{p}_i$ so that they are located as close as possible to blue points $\mathbf{q}_i$. Hence, we are interested to find out for what matrix $\mathbf{A}$ and vector $\mathbf{t}$ the function

$$\sum_i \left(\mathbf{A}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\right)^2 \tag{1.2}$$

yields the minimum, i.e.

$$\arg\min_{\mathbf{A},\mathbf{t}} \sum_i \left(\mathbf{A}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\right)^2 .$$

Since it is a sum of squares, the function is convex on its domain and thus we can find the global minimum. Firstly, we will algebraically formulate the vector of translation $\mathbf{t}_{\text{min}}$ for which the function yields the minimum. We solve the equation $\frac{\partial}{\partial \mathbf{t}} \sum_i \left(\mathbf{A}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\right)^2 = 0$.

[2]The picture originates in short computer animated film Sintel which has been made using 3D FLOSS graphics software Blender.

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{t}} \sum_i (\mathbf{A}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i)^2 &= 0 \\
2\left[\mathbf{A}\left(\sum_i \mathbf{p}_i\right) + \left(\sum_i \mathbf{t}\right) - \sum_i \mathbf{q}_i\right] &= 0 \\
\mathbf{A}\left(\sum_i \mathbf{p}_i\right) + i\mathbf{t} - \sum_i \mathbf{q}_i &= 0 \\
i\mathbf{t} &= \left(\sum_i \mathbf{q}_i\right) - \mathbf{A}\sum_i \mathbf{p}_i \\
\mathbf{t}_{\text{min}} &= \frac{\sum_i \mathbf{q}_i}{i} - \mathbf{A}\frac{\sum_i \mathbf{p}_i}{i} \\
\mathbf{t}_{\text{min}} &= \mathbf{q}_c - \mathbf{A}\mathbf{p}_c
\end{aligned}$$

Next, we will formulate the affine transformation matrix $\mathbf{A}_{\text{min}}$ for which the function yields the minimum.

We solve the equation

$$\frac{\partial}{\partial \mathbf{A}} \sum (\mathbf{A}\mathbf{p}_i - \mathbf{A}\mathbf{p}_c - \mathbf{q}_i + \mathbf{q}_c)^2 = 0.$$

For simplicity, we label $\hat{\mathbf{p}}_i = \mathbf{p}_i - \mathbf{p}_c$ and $\hat{\mathbf{q}}_i = \mathbf{q}_i - \mathbf{q}_c$.

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{A}} \sum_i (\mathbf{A}\hat{\mathbf{p}}_i - \hat{\mathbf{q}}_i)^2 &= 0 \\
2\sum_i \left[(\mathbf{A}\hat{\mathbf{p}}_i - \hat{\mathbf{q}}_i)\,\hat{\mathbf{p}}_i\right] &= 0 \\
\sum_i \left(\mathbf{A}\hat{\mathbf{p}}_i\hat{\mathbf{p}}_i^T - \hat{\mathbf{q}}_i\hat{\mathbf{p}}_i^T\right) &= 0 \\
\mathbf{A}\sum_i \left(\hat{\mathbf{p}}_i\hat{\mathbf{p}}_i^T\right) - \sum_i \left(\hat{\mathbf{q}}_i\hat{\mathbf{p}}_i^T\right) &= 0 \\
\mathbf{A}_{\text{min}} &= \sum_i \left(\hat{\mathbf{q}}_i\hat{\mathbf{p}}_i^T\right) \cdot \left(\sum_i \hat{\mathbf{p}}_i\hat{\mathbf{p}}_i^T\right)^{-1}
\end{aligned}$$

Let us remind that $\hat{\mathbf{p}}_i$ and $\hat{\mathbf{q}}_i$ are column vectors and $\hat{\mathbf{q}}_i\hat{\mathbf{p}}_i^T = \hat{\mathbf{q}}_i \otimes \hat{\mathbf{p}}_i$ is outer product of the vectors.

Now, when we obtain new locations of points $\mathbf{p}_i$ using the affine transformation of coordinates, which we label as $\mathbf{p}_i^{'}$, and simultaneously, when we deform every single pixel of the image, it is not a surprise, that we get the result illustrated in Figure 1.2.

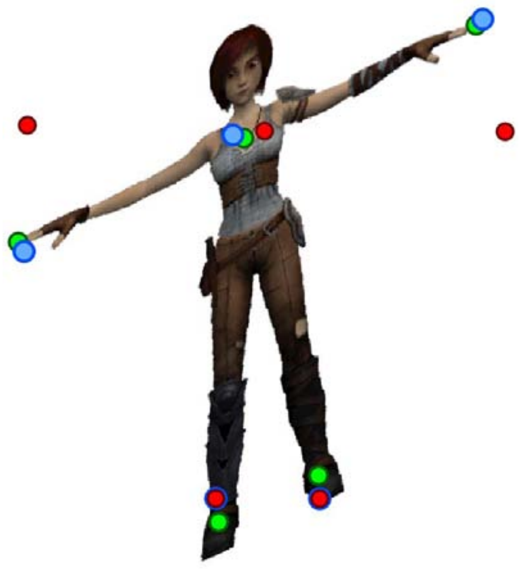


Figure 1.2: *Multipoint affine deformation of image*

1.3.2 Homogeneous coordinates

A planar point represented by the Cartesian coordinates in a form of a pair (x, y) can be expressed in homogeneous coordinates as a triple $(X, Y, w) = (xw, yw, w)$, where we usually choose $w = 1$ [12]. However, w may be generally any non-zero number. A point expressed in homogeneous coordinates as (X, Y, w) can be easily converted to Cartesian coordinates by dividing by w, that is $(\frac{X}{w}, \frac{Y}{w})$.

Affine deformation can be expressed in matrix form using homogeneous coordinates as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or alternatively as

$$\mathbf{p'}_h = \mathbf{A}\mathbf{p}_h. \tag{1.3}$$

The advantage of the matrix notation is that we can easily combine deformations using matrix multiplication. Inverse transformation can be obtained by multiplying the equation 1.3 by the inverse of the matix $\mathbf{A}$.

Affine deformation can therefore be expressed as a product of individual transformations it is composed of, i.e. as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & h_x & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or simply as

$$\mathbf{p'}_h = \mathbf{TRSHp}_h$$

where $\mathbf{p'}_h$ are the sought homogeneous coordinates of a point on the deformed image, $\mathbf{T}$ is a translation matrix, $\mathbf{R}$ is a rotation matrix, $\mathbf{S}$ is scale matrix, $\mathbf{H}$ is a shear matrix and $\mathbf{p}_h$ is homogeneous coordinates of a point on the source image.

1.4 Projective model

This model describes how a plane is deformed when viewed in space from different point of views through a pinhole camera [31]. Projective deformation can be expressed as

$$\begin{aligned} x' &= \frac{a_{11}x + a_{12}y + a_{13}}{a_{31}x + a_{32}y + 1} \\ y' &= \frac{a_{21}x + a_{22}y + a_{23}}{a_{31}x + a_{32}y + 1} \end{aligned}$$

which, in a matrix form using homogeneous coordinates, can be expressed as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or simply as

$$\mathbf{p'}_h = \mathbf{Pp}_h.$$

Projective deformation is given by four different non-colinear points and their corresponding counterparts. The matrix $\mathbf{P}$ can be acquired by solving a system of equations that can be obtained by substituting these points to the equations above. We solve the system consisting of 8 equations of 8 unknowns, which can be written in matrix form as

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

This system can be solved using Gaussian elimination method or by multiplying by inverse matrix.

A faster method is described in [12]. In a special case, when we look for a matrix of projective deformation which deforms a unit square to a quadrilateral, system of equations can be simplified. This special case can be easily generalized to a case of any square or rectangle to a quadrilateral. By performing inversion of obtained 3×3 matrix we are also able to solve the inverse problem, i.e. a quadrilateral to a square. By combining these two cases we obtain a general method for acquiring matrix of projective deformation that transforms a quadrilateral to a corresponding counterpart.

1.5 ARAP, ASAP deformation models

This section describes image deformation methods allowing to deform an image in a way that during the deformation it behaves like a real world object made of rubber. With these methods it is possible to stretch and shrink the image to certain extent and when the image is articulated (e.g. an image of a figure), it is possible to manipulate its individual joints (e.g. arms, feet). Resulting images of deformation employing these methods can look very realistically.

In this section we will describe an image deformation method employing Moving Least Squares optimization [26] and also a method described in [29] or [28] which will be further used for image registration.

All these methods respect the as-rigid-as-possible (or as-similar-as-possible) principle, i.e. to obtain a realistic result of the deformation, it is necessary to minimize the amount of shearing (and in the case of ARAP even scaling) factors involved in the deformation [26, 14, 1].

1.5.1 Deformation using Moving Least Squares

Let us now return to Figure 1.2 on page 9 and specify our task described in section 1.3.1 some more. We want the sought affine transformation of coordinates to move points $\mathbf{p}_i$ almost exactly to locations of points $\mathbf{q}_i$.

It is obvious that we are not able to achieve desired result with use of just one affine transformation. However, a combination of several different affine transformations will help us. We can achieve that by assigning a weight w_i to every pixel $\mathbf{v}$ of the image. The weight is defined as

$$w_i = \frac{1}{(\mathbf{p}_i - \mathbf{v})^{2\alpha}}, \tag{1.4}$$

thus it is a function which yields high values near points $\mathbf{p}_i$. For illustration, let us select a particular point $\mathbf{p}_j$. The nearer the pixel $\mathbf{v}$ to the point $\mathbf{p}_j$, the higher is the influence of an affine transformation that maps point $\mathbf{p}_j$ to point $\mathbf{q}_j$; α is a selected parameter.

Hence, we modify the function (1.2) being minimized to

$$\sum_i w_i \left(\mathbf{A}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\right)^2 . \tag{1.5}$$

By solving the minimizing problem symbolically, we obtain the affine transformation matrix $\mathbf{A}_{\text{min}}$ and the translation vector $\mathbf{t}_{\text{min}}$ in the following form

$$\begin{aligned} \mathbf{A}_{\text{min}} &= \sum_i \left(w_i \hat{\mathbf{q}}_i \hat{\mathbf{p}}_i^T\right) \cdot \left(\sum_i w_i \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T\right)^{-1} \\ \mathbf{t}_{\text{min}} &= \mathbf{q}_c - \mathbf{A}\mathbf{p}_c \end{aligned} \tag{1.6}$$

where

$$\mathbf{q}_c = \frac{\sum_i w_i \mathbf{q}_i}{\sum_i w_i}, \ \mathbf{p}_c = \frac{\sum_i \mathbf{p}_i}{\sum_i w_i}.$$

Since a value of weights w_i in this optimalization problem (Least Squares) vary depending on a pixel $\mathbf{v}$, this method is called Moving Least Squares optimization.

When we deform the original image using this method, we obtain the image in Figure 1.3, which safisfy our requirement, i.e. moving points $\mathbf{p}_i$ exactly to locations of points $\mathbf{q}_i$.

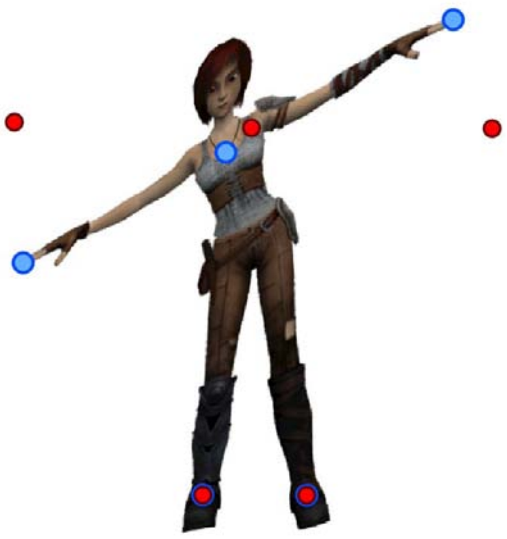


Figure 1.3: *Multipoint MLS (affine) deformation of the image*

However, the result still does not look realistically – there is a shearing visible in it. As already stated at the beginning of the section 1.5, to obtain a realistic result, the deformation has to be as-similar-as-possible or even better as-rigid-as-possible. For ASAP deformation this means that affine transformation of coordinates does not include shearing but only includes a uniform scaling, a rotation and a translation. For ARAP deformation this means that the affine transformation also does not include scaling but only includes a rotation and a translation.

Thanks to the work of Schaefer et al. [26], closed form formulas of these transformations are known (although only in 2D).

Similarity transformation matrix which is the solution of the minimization problem 1.5 is formulated as

$$\mathbf{S}_{\min} = \frac{1}{\mu} \sum_i w_i \begin{pmatrix} \hat{\mathbf{p}}_i \\ -\hat{\mathbf{p}}_i^{\perp} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{q}}_i^T & -\hat{\mathbf{q}}_i^{\perp T} \end{pmatrix} \tag{1.7}$$

where

$$\mu = \sum_i w_i \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T$$

and operator $\perp$ represents a perpendicular vector, i.e. $(x, y)^{\perp} = (y, -x)$.

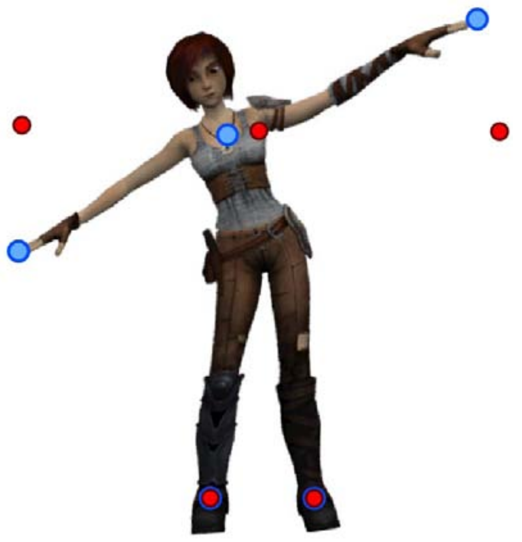


Figure 1.4: *Multipoint MLS (rigid) deformation of the image*

Rigid transformation matrix $\mathbf{R}_{\text{min}}$ is formulated identically to the similarity matrix. The only difference is in constant μ which is expressed as follows:

$$\mu = \sqrt{\left(\sum_i w_i \mathbf{q}_i \hat{\mathbf{p}}_i^T\right)^2 + \left(\sum_i w_i \mathbf{q}_i \hat{\mathbf{p}}_i^{\perp T}\right)^2} \tag{1.8}$$

After processing of our image using ARAP MLS deformation[3], we obtain a deformed image in Figure 1.4 that already satisfies also our requirements on realism.

It should be mentioned here that we obtain plaussible results only for suitable positions of points $\mathbf{p}_i$ a $\mathbf{q}_i$. The problem that appears here is caused by the measure (Euclidean distance) that is employed in the weighting function (1.4). This measure does not respect a topology of image. The consequence of this is that when we e.g. move a point located on a hand towards the body, not only the hand is deformed but also hips and the body. This can be solved by a suitable measure that respects image topology.

However, we can still find cases for which MLS deformation yields unacceptable results.

[3]This type of deformation is implemented in a package of tools for image processing Fiji as well as in the currently newest version (2.7) of graphics editor Krita. Both programs are FLOSS software.

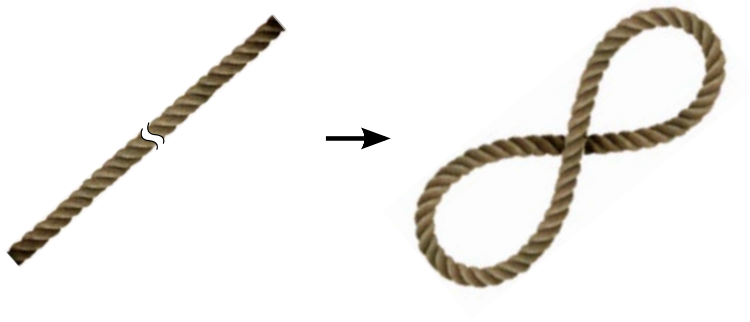


Figure 1.5: *Example of a large image deformation*

1.5.2 Coupled bodies

Now, let us look at an image of a rope and its deformation (Figure 1.5). That is one of the examples where we are not able to obtain plaussible result with MLS method.

A method [29, 28] allowing such a large realistic deformations of images will be now described. It is, generally, based on separation of a deformed object info parts that are connected with each other and their (usually rigid) transformation.

The method contains the following phases:

- construction of a lattice above an image
- deformation of the lattice
- regularization of the lattice
- transformation of the image under the lattice

Construction of a lattice above an image

In this first phase, it is necessary to construct a mesh above the source image. It can be composed of arbitrary planar shapes.

In our case we use a mesh composed of squares despite the fact that with it we are not able to precisely copy the image (as with triangular mesh). The reason is simplicity of construction of square lattice above the image (in comparison with e.g. triangulation) and lower number of lattice elements in comparison to triangular mesh. Furthermore, when we adjust the square lattice in a simple manner, we can achieve similar behaviour as with triangular mesh.

This phase takes place only at the beginning of the method.

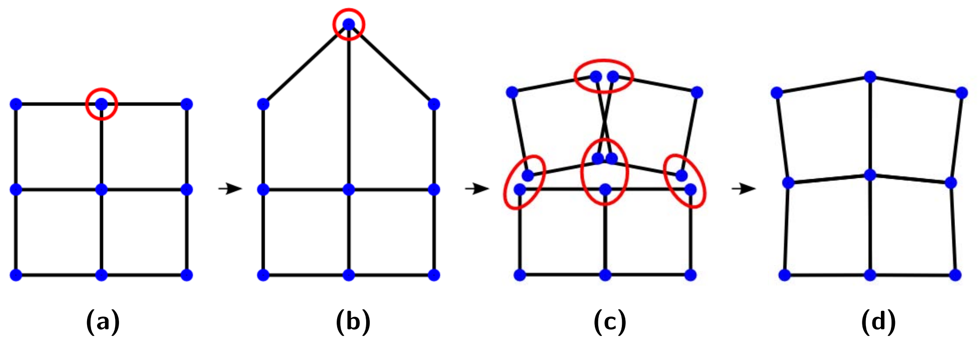


Figure 1.6: *Phases of ARAP deformation of a lattice. Figures* (**a**) *and* (**b**) *depicts moving the selected point, thus deforming the lattice. Figure* (**c**) *depicts lattice regularization and Figure* (**d**) *lattice deformed in as-rigid-as-possible manner. However, this is just one iteration of the algorithm.*

Deformation of the lattice

A lattice is actually a group of points. Identically to the section 1.3.1 on page 6, we have initial positions $\mathbf{p}_i$ of points of the lattice (or *source* or *reference* lattice) and target positions $\mathbf{q}_i$ of the lattice (or *target* or *current* lattice).

User sets new positions of some of the points $\mathbf{q}_i$ of the target lattice and thus deforms the lattice – some of the squares become quadrilaterals (see Figure 1.6b).

Regularization of the lattice

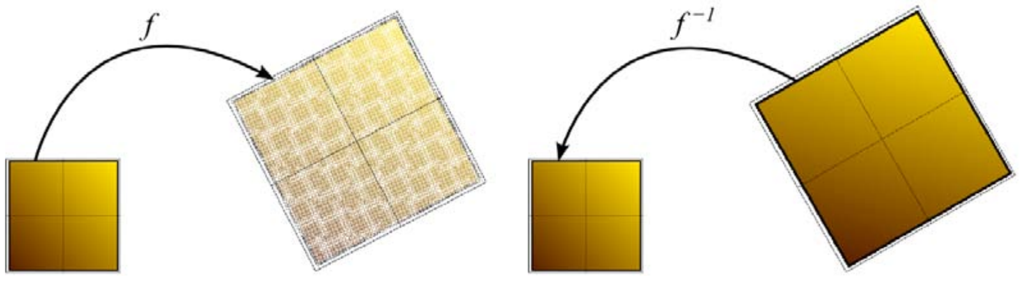
The core of the method is in this phase where we try to respect user specified constraints in form of the placed points $\mathbf{q}_i$ as well as deform the lattice in a way that every single square is deformed the most rigidly.

We perform a specified number of regularizing iterations of which every one performs the following operations:

1. Rigid (or other) transformation of every lattice square (see Figure 1.6c) using the formula 1.7, 1.8 on page 14 and $w_i = 1$.

   – By that, every quadrilateral becomes square again.

2. Centering every originally overlapping points of the lattice (see Figure 1.6d).



(a) *Forward transformation* **(b)** *Backward transformation*

Figure 1.7: *Transformation (mapping) methods*

  - This ensures that all originally overlapping points of the lattice will overlap again – and squares become quadrilaterals again.

Hundreds of these regularizing iterations are usually performed.

Projective transformation of the image under the lattice

Since we are interested in a deformation of an image and not only a lattice, we have to transform a part of the image under every square of the source lattice to corresponding squares of the target (deformed) lattice. We can solve this problem with help of projective transformation which is given, as already mentioned, by four points.

This transformation can be performed in two ways:

- **Forward transformation**
  - In this way, we apply a transformation f to every pixel $\mathbf{p} = (x, y)$ of the image under a square of the source lattice, by which we obtain coordinates of pixel $\mathbf{q} = f(\mathbf{p}) = (x', y')$ of the image under the corresponding square of the target lattice. Then we can color the pixel $\mathbf{p}$ by a color of the pixel $\mathbf{q}$.
  - This way we obtain transformed image, which usually has gaps – see Figure 1.7a. This effect occurs when transforming a smaller area to a larger one.
- **Backward transformation**
  - In this way, we apply the inverse transformation f^{-1} to every pixel $\mathbf{q} = (x', y')$ under a square of the target lattice. This gives us coordinates of pixel $\mathbf{p} = f^{-1}(\mathbf{p}) = (x, y)$ of the image under

the corresponding square of the source lattice. Then we can color the pixel **q** by a color of pixel the **p**.

– This method solves the problem with gaps – see Figure 1.7b.

1.5.3 3D ARAP, ASAP deformation

The method described in the section above can also be used for deforming 3D models in space. In comparison with the 2D case, there are several differences there. Nevertheless, the principle is preserved.

Separation of 3D model into parts

Identically to the 2D case, it is necessary to somehow separate the object which is to be deformed. There are several possible approaches here. One of them is analogical to the 2D case, that is the construction of 3D lattice composed of regular bodies (e.g. cubes) into which the model is embedded [2]. Another possible approach is utilization of a structure of the model itself which is usually composed of triangles [27]. Natural possibility is to utilize user defined bones (i.e. groups of model points that form a rigid whole) with specified weights of individual points.

Model deformation and regularization

Again as in the 2D case, user moves some of the points $\mathbf{q}_i$ and thus deforms the model.

During model regularization, we proceed in the same way as in the 2D case. However, closed form formulas (1.7 on page 13) for computation of similarity and rigidity transformations does not apply here. Nevertheless, we can still use the general formula (1.6 on page 12) for computation of affine transformation of points and obtain the similarity or rigid transformation using some method for matrix decomposition – e.g. polar decomposition [13] or singular value decomposition (SVD) [7].

To allow a greater control over the resulting deformation, it is advisable to employ a more general technique (instead of the technique used in the 2D case) to center overlapping points. The technique is called *skinning*, more specifically *linear blend skinnning* [16] or *dual-quaternion skinning* [15] of which the latter yields more realistic results.

An example of 3D ARAP deformation is depicted in Figure 1.9.

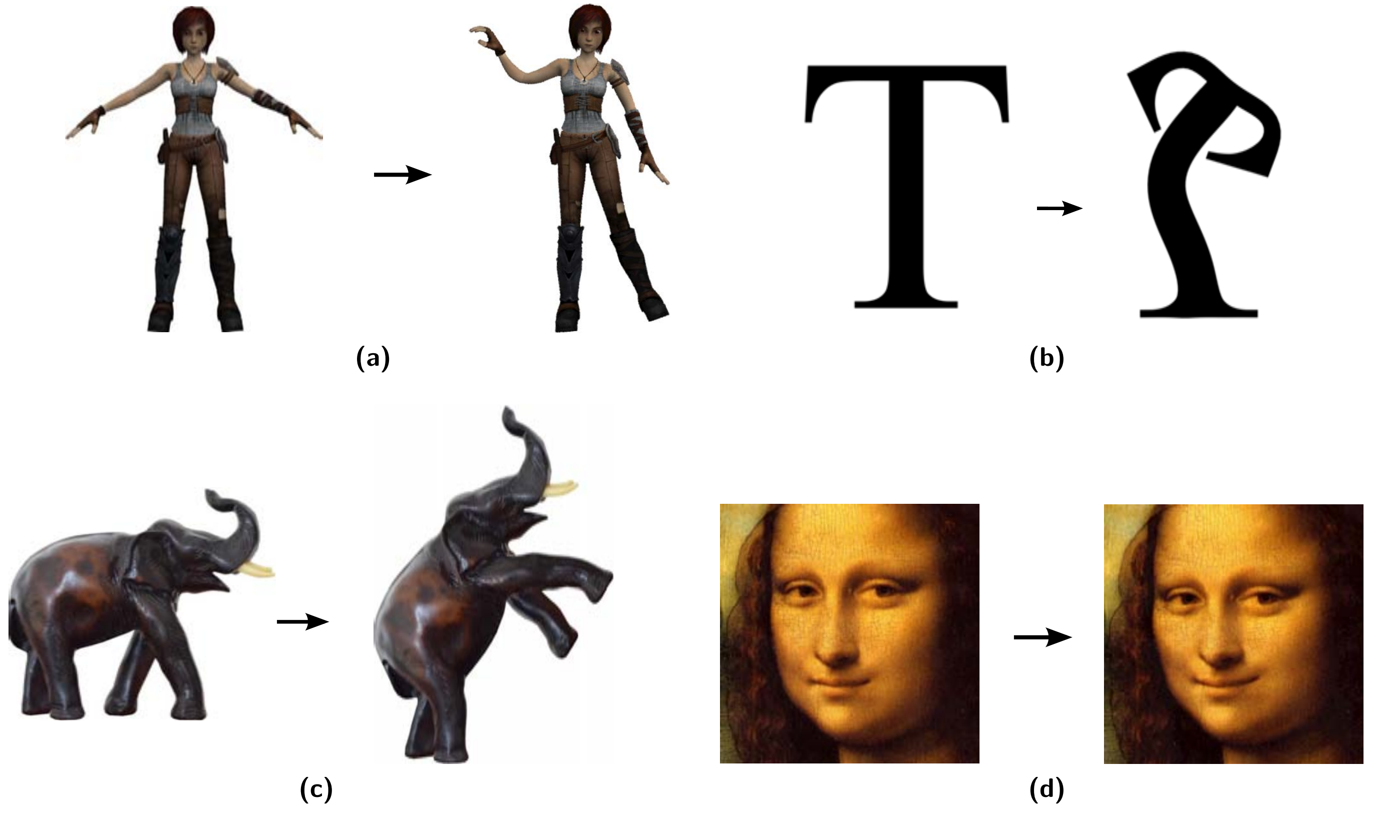


Figure 1.8: *Examples of n-point deformation employing ARAP deformation model that has been created by "n-point deformation" tool in GIMP. The implementation of this tool is described in this thesis.*

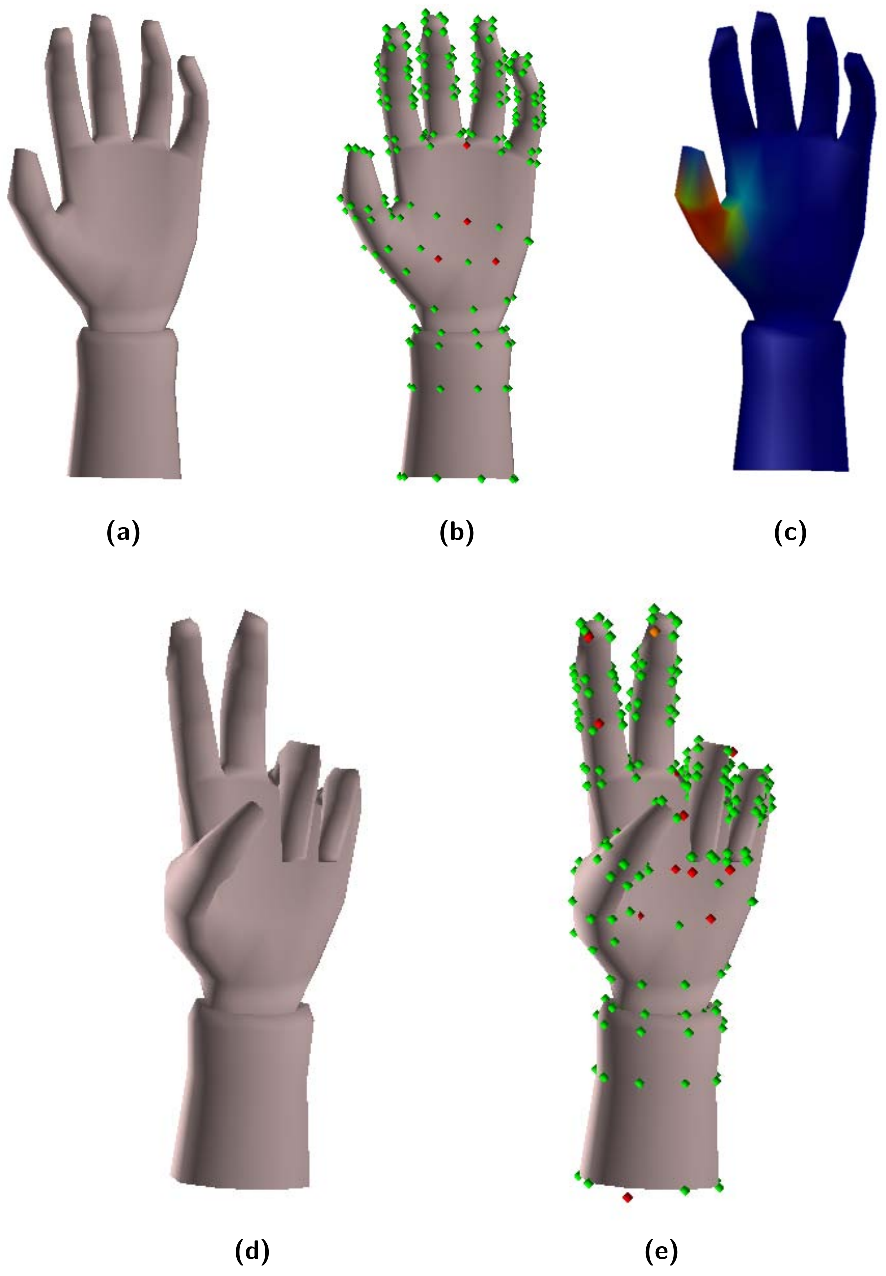


Figure 1.9: *Example of 3D ARAP deformation. Three-dimensional model of a hand in Figure* (**a**) *has been deformed, by moving several points, into a shape depicted in Figure* (**d**). *Points, the model is composed of, are depicted in Figures* (**b**) *and* (**e**). *Red points indicate fixed points. The model is composed of bones. Figure* (**c**) *depicts one of bone weights of the thumb.*

Chapter 2

Image registration

Suppose we have two images – S (source[4]) and T (target[5]). These images are somehow similar. We want to deform the image S to well align it with the image T. Problems of this type can be solved using *image registration*.

Figure 2.1a depicts one of the possible ilustrations of the problem. In this case, this is a simple task which can be solved manually by moving the image S to the appropriate position. Automation of this procedure is however necessary when we need to perform this activity often and on a large image database. More difficult task is depicted in Figure 2.1b where we need to register two or more images acquired using some of the medical imaging methods. From the picture, we can see that the sought deformation will be somehow non-linear. Another possible ilustration of applications of image registration is depicted in Figure 2.1c. There we need to use an image registration method, which utilizes some of the deformation models that preserves rigidity.

Image registration methods often somehow include *deformation model*, image *similarity measure* and *optimization method*. Some deformation models have been described in the previous chapter.

In this chapter, we will describe the basic image similarity measures and also some image registration methods with more focus on a registration method employing ARAP and ASAP deformation models that have been described in the previous chapter.

Firstly, let us look at some another applications of image registration.

[4]in some literature also floating, testing or sensed
[5]sometimes also fixed or reference

2.1 Applications of image registration

Image registration has wide range of applications and is employed in various fields (for example medical imaging, remote sensing, computer vision), as it can be used to solve the following frequently occurring problems [31, 3]:

- We have *images acquired from various points of view*, we want to compose them into a larger whole (e.g. previously mentioned panorama picture or satellite maps) or obtain a 3D representation of a scene from them.
- We have *images of the same object acquired at different times*, we want to know how the object changes in time (e.g. progress of treatment of a patient, detection of a change in security video cameras or video compression).
- We have *images acquired from different types of sensors*, the aim is to integrate different informations contained in them in order to gain more informations of the sensed scene (e.g. combination of a picture of an injury with X-ray image of it or combination of an image obtained from magnetic resonance with computed tomography image).
- We have an *image of a real scene* and we want to *find a pattern* in it for the purposes of comparation (e.g. character recognition).

2.2 Measures of image (dis)similarity

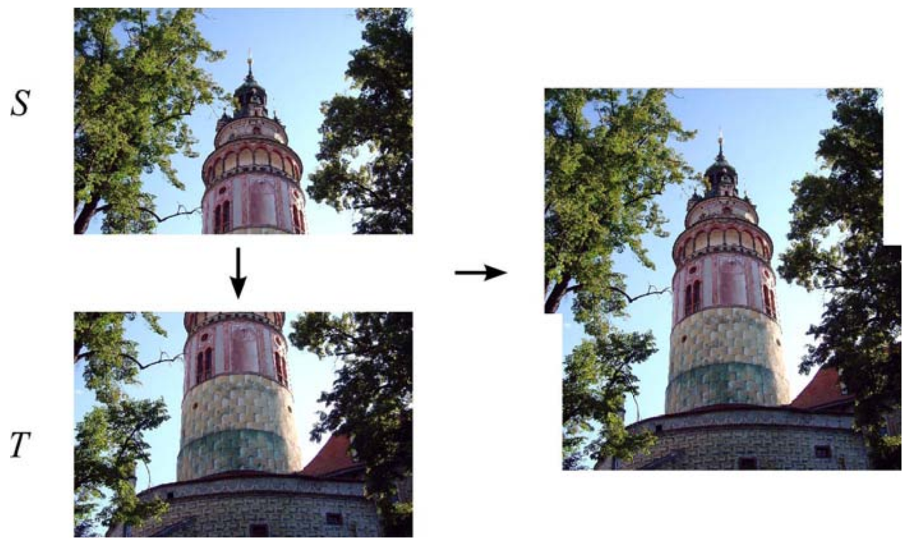
To assess the extent to which the two images S and T or their parts are similar, functions described in this section are often used. Functions which yield high values when the images are very similar are called similarity measures. Functions that have the opposite behavior are called dissimilarity measures. An extensive overview of these measures can be found in [10].

2.2.1 Sum of Absolute Differences (SAD)[6]

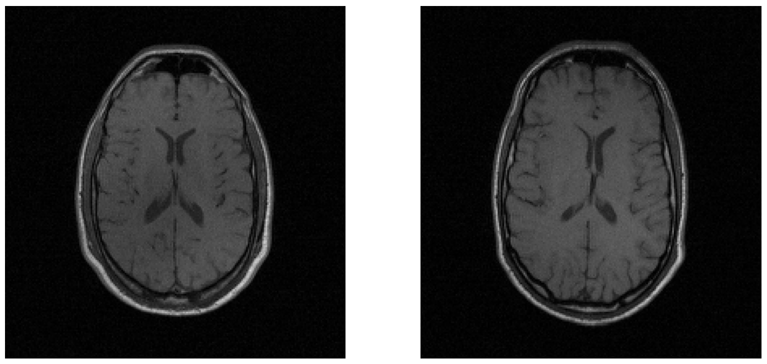
This dissimilarity measure is defined as

$$\sum_{\mathbf{p} \in N} \left| S(\mathbf{p}) - T(\mathbf{p}) \right| ,$$

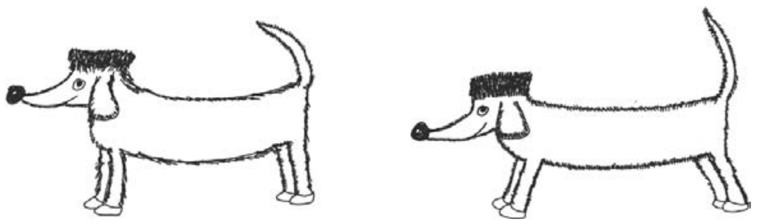
[6]or also L_1 norm or Manhattan norm



(a) *Composing images into a panorama*



(b) *Images of a transversal cut of a head of two different patients acquired using magnetic resonance for the purpose of comparison. Images come from the RIRE dataset.*



(c) *These two hand-drawn images could be two consecutive frames of an animation. Image registration can help us to e.g. create smooth transition from one frame to another.*

Figure 2.1: *Examples of some applications of image registration*

where N denotes the common area (i.e. set of pixels coordinates) of images, in which we detect similarity of the images.

The advantage of this measure is the speed of its computation. Support for accelerating the computation of this measure is incorporated in SIMD instructions in many of today's processors – namely the instruction `PSADBW`.

2.2.2 Sum of Squared Differences (SSD)[7]

SSD is a dissimilarity measure which is defined as

$$\sum_{\mathbf{p}\in N} \left(S(\mathbf{p}) - T(\mathbf{p})\right)^2 .$$

When compared with SAD, this measure is more sensitive to the greater differences between compared pixel intensities and less sensitive to additive Gaussian noise [20].

2.2.3 Normalized Cross-Correlation (NCC)

This similarity measure is defined as

$$\frac{1}{\sigma_S}\frac{1}{\sigma_T}\sum_{\mathbf{p}\in N} \left(S(\mathbf{p}) - \overline{S}\right)\cdot\left(T(\mathbf{p}) - \overline{T}\right),$$

where σ_S a σ_T are standard deviations and $\overline{S}$ a $\overline{T}$ are expected values of images intensities.

Thanks to subtraction of expected values, this measure is independent of brightness changes and thanks to division by standard deviations of intensities, it is independent of changes in contrast. Computational complexity of this measure is higher when compared to SAD and SSD.

[7]or also squared L_2 norm or squared Euclidean distance

2.3 Image registration methods

Image registration methods can be divided into area-based[8] and feature-based[9] [31].

2.3.1 Area-based methods

While estimating the deformation model, these methods work directly with the intensities of each pixel of image.

In this section, we will describe block-matching and general optimization method.

Block-matching

When the source and the target image differ only in translation (or also slight rotation), it is possible to determine the shift vector employing one of the simplest image registration methods which is a block-matching[10].

Block-matching is actually a brute-force method in which we gradually shift a part ("window") of the source image (or entire image) over the target image along the x and y axis and using selected dissimilarity measure we search for the optimum, i.e. the minimum.

Advantage of this approach is finding the global minimum, disadvantage is the time complexity of the method – $\mathcal{O}\left(|S|\,|T|\right)$, where $|S|$ is the size of the window or entire source image and $|T|$ is the size of the target image.

Block-matching method can be accelerated in various ways. One of these ways is early-termination [17]. When computing a value of dissimilarity function for the current window, we can stop the computation when the value exceeds a specified minimum. This minimum is updated whenever the lower value of dissimilarity function is found.

Optimization employing gradient descent

In the block-matching method, we gradually calculated the value of dissimilarity function for every shift of the source image over the target image and sought for a shift for which the function gave the minimal value. Analogously, we can create a function

$$E(\mathbf{t}) = d\Big(S\left(\mathbf{p}+\mathbf{t}\right),\ T\left(\mathbf{p}\right)\Big),$$

[8]or also intensity-based or pixel-based

[9]or also point-based

[10]or sometimes also template-matching

where $d(S,T)$ is a dissimilarity function representing dissimilarity measure of images S and T. Function E is often reffered to as energetic[11] function and it often contains, besides a term expressing the similarity measure, also another (regularization) terms penalizing undesirable properties of deformation – for example bending energy regularization term [21].

For example, we can use SSD dissimilarity measure and express the function E as

$$E(\mathbf{t}) = \sum_{\mathbf{p}\in N} \Big(S(\mathbf{p}+\mathbf{t}) - T(\mathbf{p})\Big)^2 .$$

To obtain a *locally* optimal shift vector $\mathbf{t}$, we can for example employ the gradient descent method [19]. Gradient descent is an iterative optimization method that proceeds at every step from the last found solution (in our example $\mathbf{t}_i$) to next solution ($\mathbf{t}_{i+1}$) in the direction opposite to gradient of the function E at the point $\mathbf{t}_i$. This can be expressed as

$$\mathbf{t}_{i+1} = \mathbf{t}_i - s \cdot \nabla E(\mathbf{t}_i),$$

where $\nabla E(\mathbf{t}_i)$ is the gradient, that means the direction of the greatest rate of increase of the function E at the point $\mathbf{t}_i$ and s is a step size.

In the same way, we can find the parameters of mapping function of other deformation models. Let us suppose tha we have a deformation model given by a function $W\left(\mathbf{p},\vec{\theta}\right)$, where $\mathbf{p}$ denotes a coordinate of a point of the source image and $\vec{\theta}$ is a vector of deformation parameters. The energy function can then be expressed as

$$E\Big(\vec{\theta}\Big) = d\Big(S\Big(W\left(\mathbf{p},\vec{\theta}\right)\Big),\ T\Big(\mathbf{p}\Big)\Big)$$

and the solution can again be found employing the gradient descent method.

Advantage of this approach is speed and disadvantage of it is that we have no guarantee of finding the global minimum. The energy function often contains a high number of local extrema and gradient descent method can thus easily get stuck in one of them.

To optimize the energy function and lower the risk of getting stuck in a local extrema, it is possible to use other optimization methods – for example, nowadays very popular evolutionary algorithms [30].

[11] or also loss, cost, utility, objective or fitness

2.3.2 Feature-based methods

When estimating the deformation model, feature-based methods use *features*, that is characteristic (or salient) objects that can be well described, are stable in time, are spread over an image and have a meaningful interpretation – e.g. bodies of water, wooded areas, populated areas, rivers, roads or crossroads (lines intersections), corners of objects in image or objects which appears as points [31].

Firstly, it is necessary to detect features in the source and the target image and then to find the most correct match of features from one image to the other.

In the past, features and their correspondences were specified manually by user. Nowadays, one of often used methods of automatic acquisition of features and their correspondences which gives good results is the SIFT method [18].

Once we have the image features and their correspondences, we can start looking for parameters of mapping function of selected deformation model. Similarly as in section 1.3.1 on page 6, we have a certain number of points located on the source image and we look for a function that maps these points to points on the target image with the smallest possible error. This problem can be solved by employing the least squares method or by employing some other method of parameter estimation [9].

2.3.3 Hierarchical approach[12]

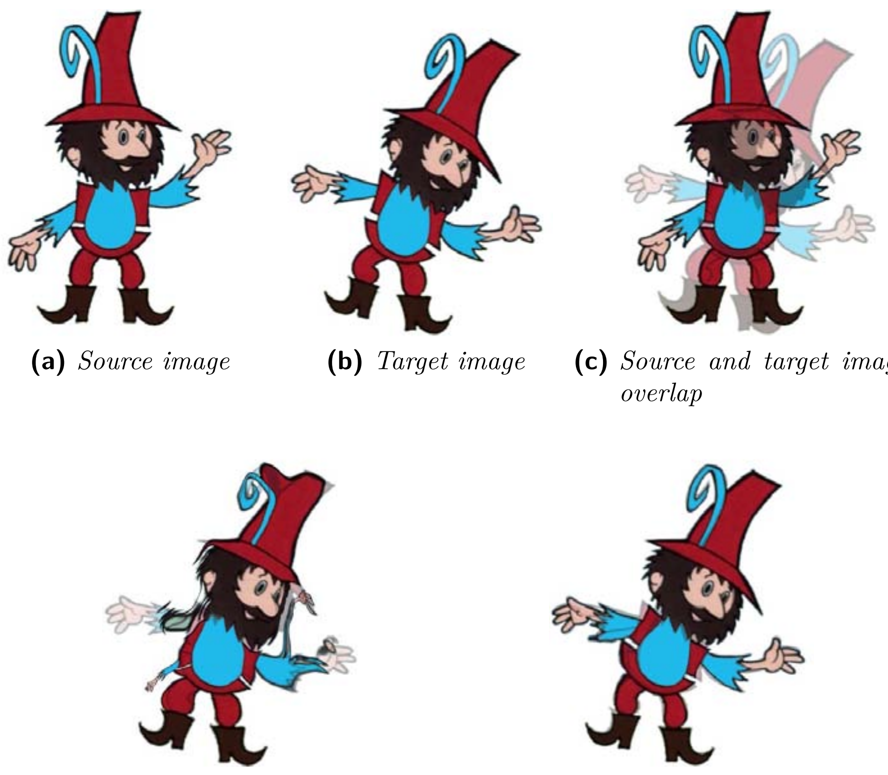
Image registration methods can be accelerated by reducing the resolution of the source and the target image. The registration is then performed on the scaled down images. This gives us a rough estimate which is then refined on images of higher resolution [8].

This procedure is often used several times. Several levels of image sizes are created while the image in nth level is half the size of the image in $(n+1)$th level. At the start, image registration is performed on the first level which gives us a rough estimate that is refined on the second level etc. The number of levels can be specified by the user or determined automatically.

The hierarchical approach can be applied also in the choice of deformation models [8]. At low resolution of the images we use simple models with low number of parameters and with every following level we use one level more complex model. Using this approach, we can get, for example, the following hierarchy of models: translation$\rightarrow$ rigid model $\rightarrow$ affine model $\rightarrow$ projective model $\rightarrow$ elastic model.

[12]alternatively also coarse-to-fine, pyramidal or multiresolution approach

2.4 Registration of hand-drawn images



(a) *Source image* **(b)** *Target image* **(c)** *Source and target image in overlap*

(d) *Result of a non-linear image registration method* [6] **(e)** *Plausible result of method [28]*

Figure 2.2: *Registration of hand-drawn images*

When registering hand-drawn images (e.g. figures), it is not possible to use feature-based methods since every single drawing is unique to some extent and hence it is not possible to find corresponding features [28]. In this situation area-based methods based on energy function optimization may lead to a success. However, if the source image and the target image differ in large non-linear deformation, the result of these methods will often not be plausible – see Figure 2.2d.

In the following sections of this chapter we will deal with image registration method [28] which gives plausible results in the mentioned situation and moreover, which has a wider range of applications.

2.5 Image registration method employing ARAP, ASAP deformation models

In ARAP registration, we use the fact that the deformation method described in section 1.5.2 on page 15 allows us to arbitrarily move points $\mathbf{q}_i$ without having to consider their mutual connection.

The actual registration is then divided into the following two phases which are continuously performed until fulfillment of specified *stop condition*.

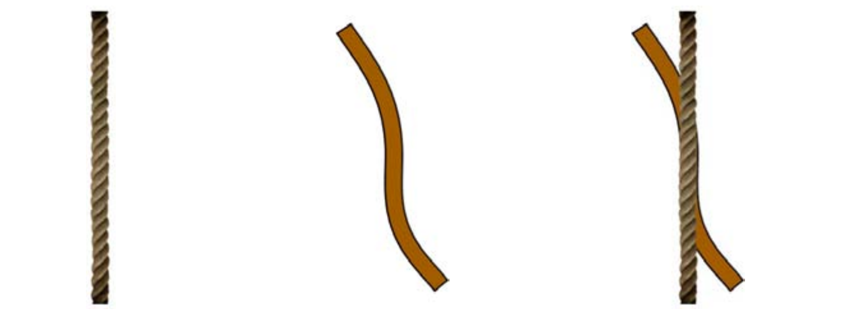
1. ”**Push**” phase, that means moving points $\mathbf{q}_i$ to *suitable locations*.
2. “**Regularization**” phase, that means performing the regularization of the lattice (or 3D model).

2.5.1 ARAP image registration[13]

In Figure 2.3c there are two overlapping objects depicted – image of a rope (the source image) and image of bent rectangle (the target image). The aim of this image registration problem is to deform the image of a rope so that it well align with the image of a bent rectangle.

In the *push* phase of ARAP registration, we move points $\mathbf{q}_i$ to suitable locations, that means locations where the area of the source image around these points differs *as little as possible* from the area of the target image (around these points). To find such a translation vector $\mathbf{t}$, we can employ the block-matching method that finds the optimal translation vector in defined search area.

[13]or alternatively n-point image registration employing ARAP deformation model



(a) *Source image* **(b)** *Target image* **(c)** *Overlapping images*

Figure 2.3: *Ilustration of a problem solved using ARAP registration*

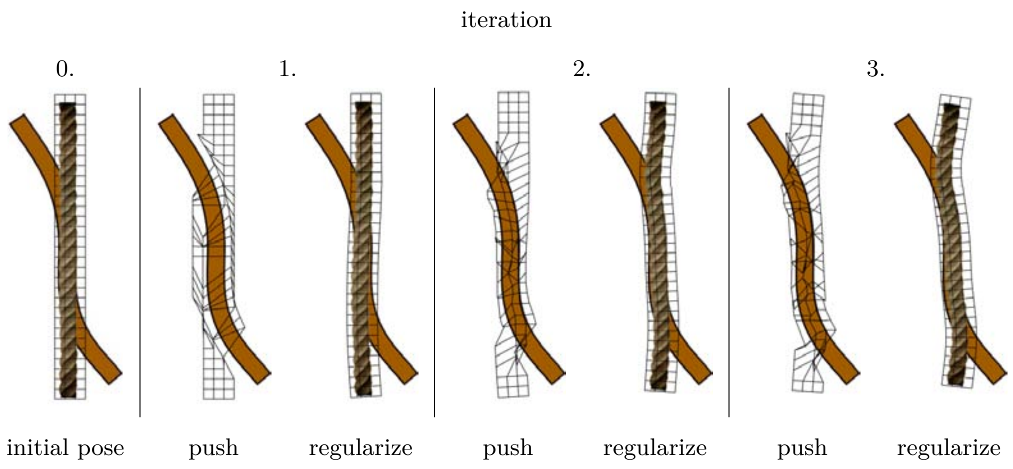


Figure 2.4: *Several first iterations of ARAP image registration algorithm*

When we use SAD dissimilarity measure to find out how well two parts of images match, the optimal translation vector $\mathbf{t}_{\text{opt}}$ can be formulated as

$$\mathbf{t}_{\text{opt}} = \arg\min_{\mathbf{t}\in M} \sum_{\mathbf{p}\in N} \left|S(\mathbf{p}+\mathbf{t}) - T(\mathbf{p})\right| , \tag{2.1}$$

where M is a search area[14] where we search for optimal shift, N is SAD „neighbourhood“, S is the source image and T is the target image.

The lattice, or more precisely its points that are shifted in the *push* phase regardless of their mutual connection, is in *regularization* phase put into a consistent state.

Figure 2.4 that shows several first iterations of ARAP image registration algorithm also shows phases of the image registration which have been described at the beginning of this section.

Stop condition

Depending on overlap of registered images and also on values of previously mentioned parameters (i.e. M, N), after several tens of iterations the source image is often deformed to well align with the target image. In case the whole process of registration is visualized and the registration is controlled by user, he can manually stop the process depending on his satisfaction with the result.

It is often useful to know an estimation of wheter the registration has reached convergence. Suitable criterion is, according to [28], monitoring

[14]Area or set of points.

of average distance (d_{avg}) of points $\mathbf{q}_i$ of the deformed lattice from points of initial pose lattice (or points $\mathbf{p}_i$ of the reference lattice). The average distance can be formulated as

$$d_{\text{avg}} = \frac{1}{|P|} \sum_i \|\mathbf{q}_i - \mathbf{p}_i\| \,, \tag{2.2}$$

where $|P|$ is the number of lattice points.

The average distance has increasing character and after some number of iterations it stabilizes with a tolerance at certain maximal value. We can stop the registration after a fixed number of iterations in which this distance does not change (with a tolerance).

Problems of this image registration method

As any method even this one has its limitations. The method does not lead to plausible results in the following cases:

– unsufficient overlap of source and target images

– the source and the target image differs in too large amount of deformation

2.5.2 3D ARAP registration

Let us mention a problem from „markerless motion capture“ (or „markerless“ MoCap) field, which deals with capturing of motion of real objects, mostly persons, without the need of wearing special markers. Usually, the aim of MoCap is to put a 3D model into a motion based on the captured one. For solving this problem, ARAP registration method can be employed.

A scene after which the actor moves, is captured by several calibrated cameras. Sequences of images recorded by individual cameras are then preprocessed – e.g. background is removed, a part of image containing the actor is modified to colour match with the 3D model. The model should match as closely as possible with the actor.

At the beginning of the recorded sequence, the actor typically stands in T pose. Before the actual registration it is necessary to place the 3D model in space as accurately as possible according to images of actor from individual cameras.

During registration of the 3D model onto pre-processed images from individual cameras, we can use similar procedure as in image registration that has been described in section 2.5.1. Let us suppose that we have i

cameras. To illustrate the principle, we now focus only on the first image from sequences of individual cameras.

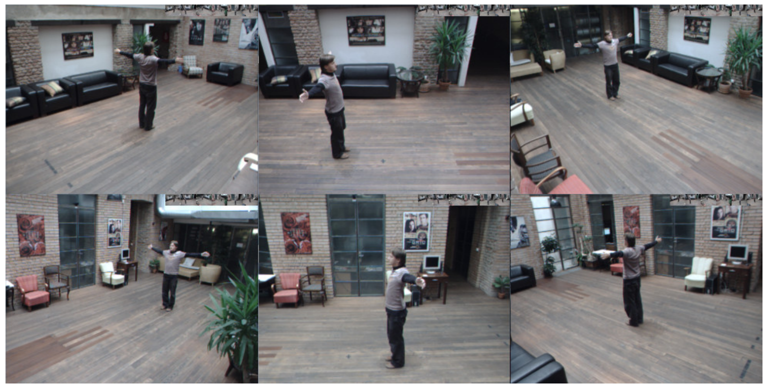
In *push* phase of 3D registration, we need to move every point of the 3D model to a suitable location. Firstly, we render the 3D model from a view of the first (virtual) camera. Employing the block-matching method, we perform registration of the rendered image (i.e. the source image) onto the image from the camera (the target image). We perform this procedure for every camera and thus we find i vectors of translation in image planes of individual cameras[15] for every point of the 3D model. We transform each of i 2D translation vectors of one point of the model to world coordinates and thus we get 3D vectors. We compute average translation vector in space and then we shift the point about it. We perform this procedure on every point of the model. Then we perform a regularization of the 3D model. By this, we described one iteration of this 3D registration.

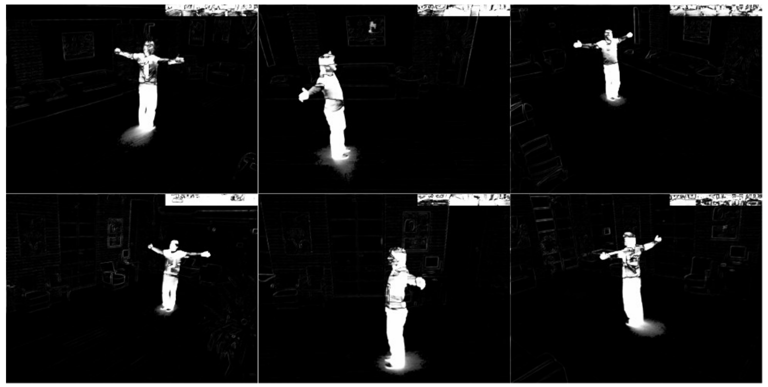
2.5.3 Interactive registration

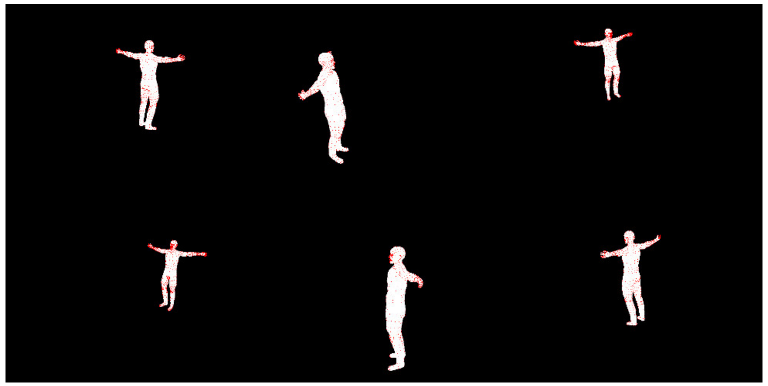
Thanks to the fact that the deformation method employed in the registration allows us to arbitrarily move points without having to consider their mutual connection and also to the fact that the registration method is iterative, we can let user influence the registration process.

The user can watch the progress of registration and in situation when the registration method has a problem with aligning the source with the target image or object (see e.g. problems of this image registration method on the previous page), he can help it by moving one or several points.

[15]That means, in individual views from virtual cameras to 3D model.


(a) *Pictures of actor standing in T pose captured using six calibrated cameras*


(b) *Pictures after preprocessing (removal of background, modifying of colors)*


(c) *Pictures of 3D model properly positioned on scene*

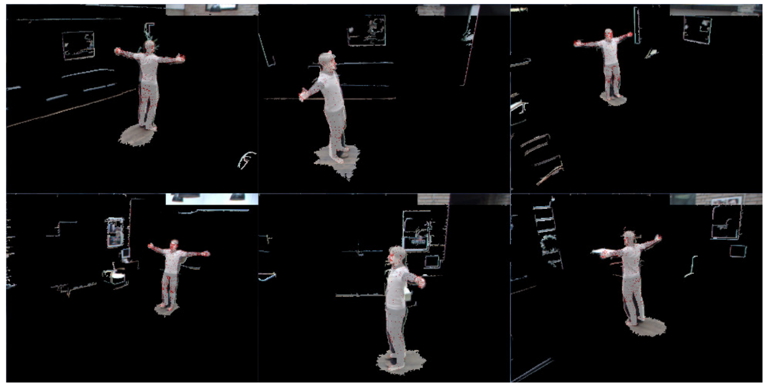

(d) *Overlap of pictures of 3D model with pictures of actor*

Figure 2.5: *Example of preparation for 3D ARAP registration employing blockmatching. These images come from a videoclip created at Universal Production Partners Ltd.*

Chapter 3

Implementation

One of the goals of this thesis was to implement the ARAP/ASAP image registration method described in section 2.5.1 and integrate it into the graphics editor GIMP. Since GIMP is written in the C programming language, the choice of programming language for the implementation of this registration method was obvious.

The implementation proceeded as follows. First, the standalone ARAP n-point deformation algorithm, that was described in section 1.5.2, was implemented. For easier debugging of the algorithm, GUI was created using Allegro 5 graphics/gaming library. Subsequently an *operation* allowing to perform n-point image deformation was implemented into GEGL library. This library is employed in GIMP. In order to use one and the same implementation of the algorithm with Allegro library, GEGL library and also with some other library or program that allows graphical ouput, a library for n-point deformation was created. After that, n-point deformation tool was implemented into GIMP. The tool formed a base for n-point registration tool which was implemented soon afterwards. The n-point deformation tool was created during a three-month long work within the Google Summer of Code 2013.

This chapter contains, besides a description of implementation details, a justification of some implementation decisions and a description of user interface of developed tools. Besides that, current trends of GIMP development are described here as well.

3.1 Implementation of the algorithm in C

A shared (dynamic) library `libnpd` was created. The library contains data types and functions allowing to perform n-point image deformation and registration. The library is designed in a way so that it can be used with various graphic apparatus (that is for example graphics library) and thus requires to implement some graphics functions (e.g. `get_pixel_color`, `set_pixel_color`) and data types (image, display).

In this section, we will describe data types and parts the library is composed of, that means the computational part, the graphics part and other routines.

3.1.1 Data types

The library defines some transparent and some opaque data types. Opaque data type is a data type which is incompletely defined for some part of a program. Data type which is completely defined is called transparent. In C language, we can manipulate with opaque type only using pointers. Only parts of a program which knows the implementation of this type, and thus for which this type is transparent, can access its data. Let us give some examples using `libnpd` library. Every data type that is described here is in the form of a record (that is `struct` in C language).

Opaque data types

The library defines opaque data type **Image (NPDImage)** which is a record that represents an image. The record can be composed of image data, image width, height and so on. The library accesses the image data of the image using ”opaque” functions as `get_pixel_color` and `set_pixel_color`. These functions are defined as pointers to functions. How these functions are implemented depends on a user of the library.

Another opaque data type that is defined by the library is **Display (NPDDisplay)**. This type represents a display apparatus or more precisely a graphics output. Implementation of this type can contain for example a pointer to a graphics context (e.g. data type `ALLEGRO_GRAPHICS` in Allegro library or `cairo_t` in graphics library Cairo) or a pointer to an image into which the result of deformation will be rendered. Concrete implementation of this data type depends on user of the library.

Transparent data types

The library defines the following transparent types:

– **Point (NPDPoint)** – This type represents a point with coordinates (x, y) which is used for deformation purposes. In addition to that it contains a weight w and another informations that are useful for purposes of deformation.

– **Bone**[16] **(NPDBone)** – This type represents a group of points forming a rigid whole. In context of this n-point image deformation, this "bone" represents a square. However, it can also represent some other shape – for example a triangle in case we use triangle mesh instead of a square lattice constructed above image.

– **Hidden Model (NPDHiddenModel)** – This type holds the source and the deformed lattice (or in other words, "reference" and "current" group of bones), a list of overlapping points (see below) and parameters of the deformation in one place together. The type (struct) contains only fileds that are not tied to any method of displaying the model (hence the name "hidden model").

– **Cluster of Overlapping Points (NPDOverlappingPoints)** – For the purposes of deformation is useful to remember which vertices (points) of individual squares in the source lattice overlap. We often need to manipulate every point in a cluster somehow – for example we need to move every point in a cluster to certain position or to set equal weight to every point of a cluster. Points in a cluster are called *overlapping points*.

– **Control Point (NPDControlPoint)** – When deforming a lattice, we can manipulate individual clusters of overlapping points – e.g. we can move a cluster to a new position $\mathbf{p} = (x, y)$. After mesh regularization, the cluster moves to a position that is given by a centroid of cluster points. That can be undesirable. The user, for example, requires the cluster to be fixed at the position $\mathbf{p}$. This can be solved by not transforming cluster points during the regularization. A slightly different solution is offered by this data type which remembers user defined position and its assignment to a cluster. Before every deformation iteration, according to positions of control points the positions of associated *overlapping points* are set.

[16]The name "bone" was borrowed from the field of 3D graphics, where this term often refers to a group of points (in space). At the time of implementation of the n-point deformation algorithm, algorithm data types were designed in a way so that it can also be used (with minor modifications) for the 3D version of the n-point deformation.

- **Model (NPDModel)** – This type holds the hidden model, the defined control points, the source image, display and parameters of deformation. It is actually a lattice tied to an image.
- **Color (NPDColor)** – This type represents a color composed of four eight-bit color components (R, G, B, A).

3.1.2 Computational part of the library

The computational part of the library provides deformation of the lattice (i.e. of the hidden model). Among other, this part of the library contains functions that allow

- to perform specified number of deformation iterations and thus to deform the lattice according to the principle described in section 1.5.2,
- to peform "push" phase of the registration according to the principle described in section 2.5.1.

3.1.3 Graphics part of the library

Graphics part of the library contains functions that are independent on graphics apparatus, that is for example graphics library. Among other, this part of the library contains functions that allow

- to create a model, i.e. to create square lattice with specified square size above specified image,
- to draw (deformed) model into an image,
- to draw model's lattice.

When used with a graphics library it is necessary to implement `NPDImage` and/or `NPDDisplay` data types and also to implement the following two functions:

- Function `npd_get_pixel_color (NPDImage *image, gint x, gint y, NPDColor *color)`, that is a function that retrieves a color of a pixel at position (x, y) in specified image and stores it into a variable pointed to by parameter `color`.
- Function `npd_set_pixel_color (NPDImage *image, gint x, gint y, NPDColor *color)`, that is a function that stores a color pointed by parameter `color` into a pixel at position (x, y) in specified image.

The library uses its own functions for bilinear interpolation and alpha-blending. For obtaining samples from an image it uses the aforementioned `get_pixel_color` function. When we want to use a better interpolation technique we can implement the following function instead of the aforementioned two:

– Function `npd_process_color (NPDImage *source_image, gfloat ix, gfloat iy, NPDImage *deformed_image, gfloat ox, gfloat oy)`, that is a function that obtains the source and the deformed image and that is expected to process (in a sense of interpolation and alpha-blending) a pixel at position (ix, iy) in the source image and to store the result into a pixel at position (ox, oy) in the deformed image.

To allow drawing the model's lattice it is necessary to implement a function

– `npd_draw_line (NPDDisplay *display, gfloat x0, gfloat y0, gfloat x1, gfloat y1)`, that is a function that draws a segment given by points (x_0, y_0) a (x_1, y_1) on defined display.

3.1.4 Other rutines implemented in the library

The library also contains common functions for manipulation with deformation – for example functions for model initialization/freeing and functions for control point manipulation.

3.2 Implementation into GIMP

3.2.1 GIMP and current development trends

GIMP is a graphics software used most often for the purposes of creating and editing raster graphics. It is a FLOSS software released under GNU GPL. The program is multi-platform – with each new stable version, a version for Linux (or other Unix-like systems), Windows and Mac OS X is released. The current stable version of GIMP is of the 2.8 series.

GIMP is written mainly in C programming language (more precisely C89[17]). The C language itself is not object-oriented. When creating large projects, the use of object-oriented approach is advantageous. GIMP uses

[17]This standard is fully supported by many compilers and a portability to other systems is therefore much easier.

the C language enriched with object-oriented approach using GObject object system, which is part of GLib library.

Development of GIMP was started by Spencer Kimball and Peter Mattis in 1995. After several years, the development was fully in the hands of the community which had meanwhile formed around this program. Today, the program can be used for both amateur and professional use. Gimp is often criticized for two reasons – it cannot work with high depth color images and it does not allow some non-destructive image editing techniques. These drawbacks should be resolved with help of GEGL library, which is used in current versions of the program very intensively.

GEGL and BABL libraries

GEGL is a graphics library which uses specified oriented acyclic graph to process images. This graph is made up of individual nodes that can represent graphics operations as well as another graph. Edges that connect individual nodes determine the order in which the graph will be processed.

The library currently contains a large amount of operations allowing, for example, to load (and save) graphic files of various graphic formats, to generate image (e.g. fractals), to use various graphic filters and to create compositions using Porter-Duff compositing operators [23]. It is not hard to implement another operations into the library.

The library is ready to work with different pixel formats (with different bit depth, color model, etc.). For this purposes, it employs BABL library. The BABL library is being created along with GEGL library and is specialized in conversion between various formats.

Although the motivation for creating the GEGL library was its employment in GIMP, it can be easily employed within another aplications – it is, for example, currently employed in the development version of a painting application MyPaint.

The first stable version of GIMP which offered an option to take advantage of some of the benefits of the GEGL library was of the 2.6 series. It is planned for the oncomming stable release of the 2.10 series to have all obsolete code replaced by code that uses GEGL and BABL libraries – this is one of the things developers currently work on.

GIMP plug-ins

Functionality of GIMP can be extended using plug-ins. They can be written in various programming languages – namely Scheme, Python, Perl and C. Plug-ins written in C are compiled and should be faster than the plug-ins written in other languages which are interpreted. There are several tutorials which explain in detail the process of writing a plug-in. Thanks to these tutorials[18], the process of writing a plug-in is much easier. Most of these tutorials were written at a time of the old GIMP core (which means they do not use GEGL library) and so it is necessary to avoid using deprecated functions for accessing individual pixels of image.

GIMP incorporates a large amount of plug-ins, the most of them function as graphic filters. The current development trend is not to create a graphic filters in form of GIMP plug-ins, but instead to create GEGL operations within GEGL library. This also applies to some other types of plugins – for example plug-ins for loading (and saving) images of various graphic formats. Such an approach has several advantages. For example it is usually not necessary to create separate GUI for GEGL operation in GIMP – all GEGL filters can share similar GUI. Preview of result of such graphic operation is displayed directly on the canvas and not in a small preview window (as it was used to be earlier). Another advantage is the possibility to use the graphic operation in another graphics software. This can reduce duplication of the same functionality between various graphic FLOSS software and thus it can support a pressure on the optimization of these functionalities. Classic way of creating GIMP plug-in has to be selected in situation when a plug-in containing e.g. specific GUI has to be created.

3.2.2 GEGL operation implementing n-point deformation

N-point deformation algorithm was implemented into the GEGL library as a new operation. This operation employs `libnpd` library which was described in section 3.1. The `libnpd` library is now a part of a development version of GEGL library. An example of how it is possible to connect the `libnpd` library with a graphic apparatus, that is the GEGL library, is contained in the source code of the GEGL operation which is called NPD.

3.2.3 N-point image deformation tool

When designing the implementation of n-point image deformation into GIMP, two variants were considered – implementation of a plug-in or internal tool. Let us discuss now the pros and cons of both variants.

[18]Tutorial which describes the process of writing a plug-in in C language is available at `http://developer.gimp.org/writing-a-plug-in/1/`.

Discussion of implementation variants

Distribution of a plug-in does not depend on "GIMP release periods". When there are some enhancements or bug fixes in a plug-in, a new version of a plug-in can be released almost immediately. For the internal tool, the situation is worse for the user. A new version of the tool is firstly available in the development version of GIMP and it may take quite a long time until a new stable or bugfixing version of GIMP is released and available to user. User can build the development version himself or there exists so called nightly-builds[19], but for a lot of users this process is too complicated. In some situations, there can be long intervals between the releases of bugfixing versions.

The need to manually install the plug-in can be taken as a disadvantage of this variant – it is inconvenient for the user. When distributing the plug-in separately from GIMP, the plug-in is not so "visible" and can attract smaller group of potential users. This may be related to lower efforts of other developers to improve the plug-in and update it for new versions of GIMP without requiring the original author of the plug-in to do that himself. However, if the plug-in could be useful for a large number of GIMP users, there is definitely an option to negotiate its inclusion in the official distribution of GIMP.

Writing a plug-in is not as demanding as creating internal tool, where a programmer which is not familiar with GIMP development needs to study new techniques (for example object-oriented approach using GObject object system) and to understand GIMP internals. It is also appropriate (and for the inclusion of the code to the main branch it is required) to follow certain style of writing a code – GIMP uses a coding style similar to GNU coding style. As already mentioned, there are tutorials explaining the process of writing a plug-in. There is considerably less informations on how to write internal tools[20]. More information can be drawn from existing source code and comments contained in them, from comments contained in individual commits in versioning system (Git) or ask GIMP developers for advices (mostly through the IRC channel or mailing list).

GIMP plug-in is a program that is run by GIMP and that employs some of GIMP's functions. If plug-in creates its own GUI, the interface is not part of GIMP's GUI (it is a separate window with plug-in's GUI). Therefore it is not, for example, possible to create elements of plug-in's GUI on GIMP's canvas. With internal tool, this is, of course, possible. Plug-in written

[19]That means at night automatically built version of the program based on the most up-to-date source files of the program.

[20]Although a nice tutorial is available in [4].

in C language can access GIMP core's functions only using GIMP library (LIBGIMP). With internal tool, possibilities regarding this are wider.

N-point image deformation as internal tool

The variant of implementation of internal tool was picked especially for the following reasons:

- GIMP does not contain a tool allowing *as-rigid-as-possible* image deformation, however, it has a tool named Cage Tool which allows to deform an image using a cage. The result of this tool is not as-rigid-as-possible and thus it is more difficult to obtain realistic deformations of images capturing real world objects. From user perspective, the process of deformation using Cage Tool is relatively cumbersome since at first user has to manually create a cage surrounding the deformed object and only after that he can start deforming the cage using points it is composed of. In videoclip contained in enclosed medium, there is an ilustration of how quickly a user is able to deform an image of a figure using n-point deformation tool in comparison with Cage Tool.

- According to the high number of positive responses to an intention to implement n-point deformation tool[21] it can be concluded that this tool can be useful for a large group of GIMP users. This argument can be also supported by the fact that the proprietary graphics software Adobe Photoshop since version CS5 includes an internal tool allowing ARAP image deformation as well.

- From the user perspective, it is more convenient to work with GUI, which is seamlessly integrated in GIMP. In our case, individual control points (handles) can be drawn directly onto the canvas. During the deformation process, user can e.g. easily zoom a certain part of image that is being deformed and focus on details or he can arbitrarily rotate the canvas.

- The proposal to implement the internal tool was accepted by GIMP developers and a substantial part was implemented during Google Summer of Code 2013.

[21]When the idea to implement n-point deformation into GIMP appeared, analysis of interest in this feature was carried out. At the beginning, a blog post describing the intension was created. It was commented by several supporting comments. The proposal was later sent to the GIMP Developer mail list and thus expanded through social networks. Videoclips presenting the proposed tool had over 35 000 views and over 500 positive comments until December 2013.

- There is a possibility to use the tool as a basis for implementing n-point image registration into GIMP.

Implementation details

Every tool in GIMP is implemented as a class extending a parent class named `GimpTool`. This parent class provides the basic functionality common to all tools in GIMP. In particular, the functionality is represented by methods called when

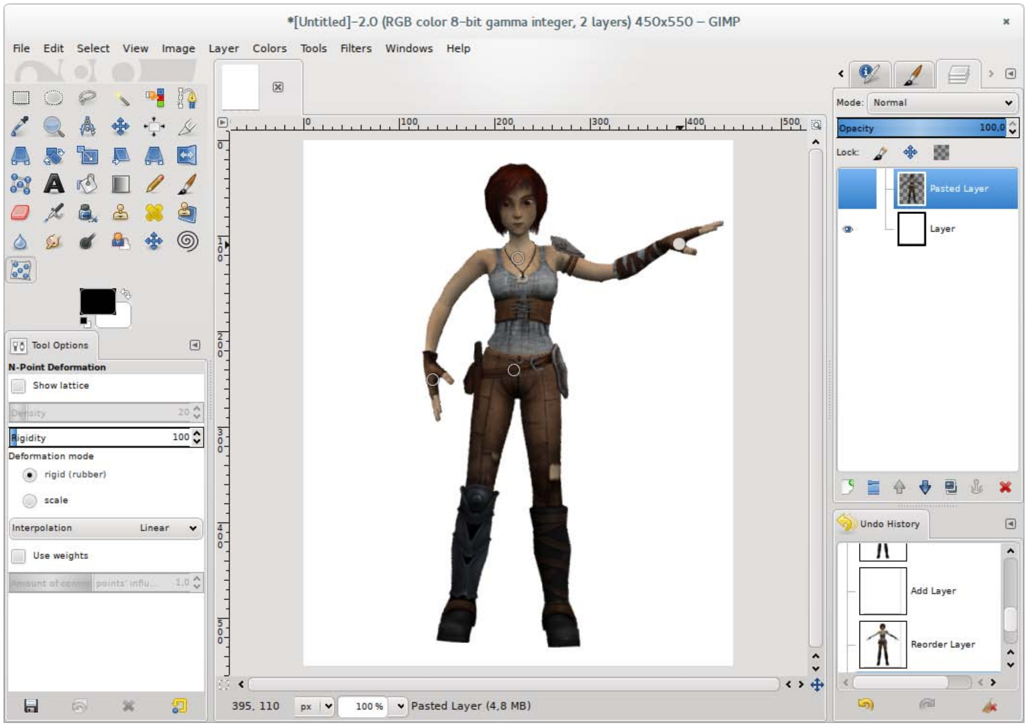
- there are keyboard or mouse events (that is a keypress, mouse move or mouse drag and drop etc.),
- user changes tool's settings using GUI,
- a tool is paused, resumed or stopped,
- undo/redo is performed

and by some another methods[22]. The mentioned (and another) methods can be overriden in a subclass.

The aforementioned class is extended by class named `GimpDrawTool` that allows its descendant classes to add GUI elements onto canvas and that is able to draw these elements. These elements include control points (handles), basic plane shapes, guide lines, paths, text cursor and also a live preview of a result of operation that is performed by a tool. This class is extended by various classes representing tools, let us mention for example a group of painting tools, selection tools, transformation tools and also a tool for writing a text.

The class `GimpDrawTool` is also extended by a class named `GimpNPoint-DeformationTool` that implements the n-point deformation tool. This class employs `libnpd` library. Using (deformation) thread, it performs a deformation of an image, using (preview) thread, it draws at regular intervals a preview of current state of the deformation. The preview thread calls the methods of `GimpDrawTool` in order to redraw GIMP's GUI. Every GIMP tool can define its own set of settings and their graphic representation within GIMP tool's GUI. For these purposes, a class named `GimpToolOptions` is employed. This class is inherited by individual tools (or more precisely by classes describing individual tools options). N-point deformation tool employs its own class named `GimpNPointDeformationOptions`.

[22]A list of all available methods of the class `GimpTool` is located at `GIMPSRC/app/tools/gimptool.h`, where `GIMPSRC` is a path to a directory with GIMP source code.



File Edit Select View Image Layer Colors Tools Filters Windows Help
Tool Options
Undo History
Add Layer
Reorder Layer

*[Untitled]-2.0 (RGB color 8-bit gamma integer, 2 layers) 450x550 – GIMP

N-Point Deformation
Show lattice
Density 20
Rigidity 100
Deformation mode
[✓ rigid (rubber)]
scale
Interpolation Linear
Use weights
Amount of control points' influ... 1.0

Mode: Normal
Opacity 100.0
Pasted Layer
Layer

395, 110 px 100 % Pasted Layer (4,8 MB)

Figure 3.1: *N-point image deformation tool in development version (2.9) of GIMP*

Description of user interface

Figure 3.1 depicts how the n-point image deformation tool is integrated into GIMP's GUI.

User can activate the tool using an icon located in the toolbar or alternatively using an item in the main menu or using a keyboard shortcut. However, this does not start the tool. GIMP allows user to work with multiple images. It is possible to switch among them or have them displayed side by side. In order to allow a tool to get access to image data of a particular image, it is necessary to click on the image. This will start the tool.

After starting the tool, an automatically constructed lattice composed of squares of the specified size apperas over the image. The individual lattice squares are constructed only over parts of the image which contains at least one visible pixel. When the image does not contain an alpha channel, the lattice is constructed over the whole image.

With a mouse click it is possible to place control points over the image and with dragging them it is possible to deform it. As in other GIMP tools user can select (and deselect) one or more control points using the SHIFT key or using so called rubber band selection. Selected control points can be deleted using the DELETE key. The last added point can be removed using the BACKSPACE key. By holding this key it is possible to quickly remove all of the control points.

The behaviour of the tool can be adjusted using the following set of settings which are depicted in Figure 3.1 at the bottom left corner:

- An option to show or hide the lattice.
- An option to specify the size of lattice squares – only before starting the tool.
- An option to specify deformation rigidity, that is the number of deformation iterations performed prior to every rendering of the model (i.e. deformed image).
- An option to specify the deformation type (i.e. deformation model) – there are two possibilities: rigid (ARAP) or scale (ASAP)
  - "Rigid" type is suitable for bending objects or manipulating parts of articulated objects. It does not give very good results in greater stretching or shrinking of an object (see a videoclip on enclosed medium).
  - "Scale" type uses ASAP deformation model and thus is able to scale some parts of the image (see a videoclip on enclosed medium).
- An option to employ weights that comes from Moving Least Squares image deformation and to set a parameter α (see section 1.5.1).
  - This option, while using a rigid type of deformation, gives better results in greater stretching or shrinking of an object – for example, it is very useful for a face deformation (see a videoclip on enclosed medium).
- An option to specify a method for interpolation of the resulting image (for a preview of the deformation, bilinear interpolation is used) – user can choose from all of the methods that are available in GIMP (or more precisely in GEGL library) – i.e. nearest neighbour, bilinear, cubic, NoHalo and LoHalo interpolation method.

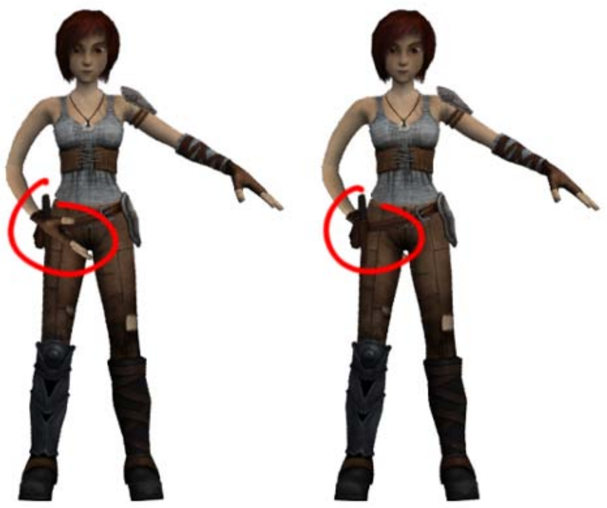


Figure 3.2: *Depth of control points*

If the user is satisfied with the result of the deformation, he can stop the tool by pressing ENTER. The deformed image is then redrawn using the selected interpolation method. If the user wants to stop the tool and discard changes in the image, he can press ESC.

Current problems of the tool

The tool has currently problems regarding speed when working with large images, mainly due to redrawing the preview of the deformation. The preview has to be rendered several times per second, which is what causes the problems. The solution is to deform a scaled down version of the image during the preview and to render only parts of the image that are visible in the GIMP window.

Currently, user does not have an option to set a depth of individual control points and thus he cannot specify which part of the overlapping lattice (image) will be visible. For example, we have an image of a figure, we move its hand so that it overlaps its body. We want the hand to be in front of the body (to be visible) or hidden behind the body – see Figure 3.2.

The tool currently does not use multiple CPU cores to speed up its computations.

3.2.4 N-point image registration tool

As the n-point deformation, the n-point image registration was implemented into GIMP as an internal tool. The main reasons for this decision included the possibility to use existing implementation of n-point deformation tool and the possibility to allow the user to easily affect the registration process (see section 2.5.3).

Implementation details

A class named `GimpNPointRegistrationTool` which extends `GimpNPoint-DeformationTool` class was created. The n-point deformation tool class was modified to allow its employment for other purposes – e.g. for registration purposes. As previously mentioned, the n-point deformation tool uses a thread to perform the deformation. Within this thread a method, the deformation code was moved into, is called. This method is overriden in `GimpNPointRegistrationTool` by a method performing registration described in section 2.5.1.

A class `GimpNPointRegistrationOptions` defining a set of settings of the tool was created. This class extends `GimpNPointDeformationOptions` class which was modified to allow its subclasses to use set of generic settings of the deformation (i.e. a set of settings excluding MLS weights).

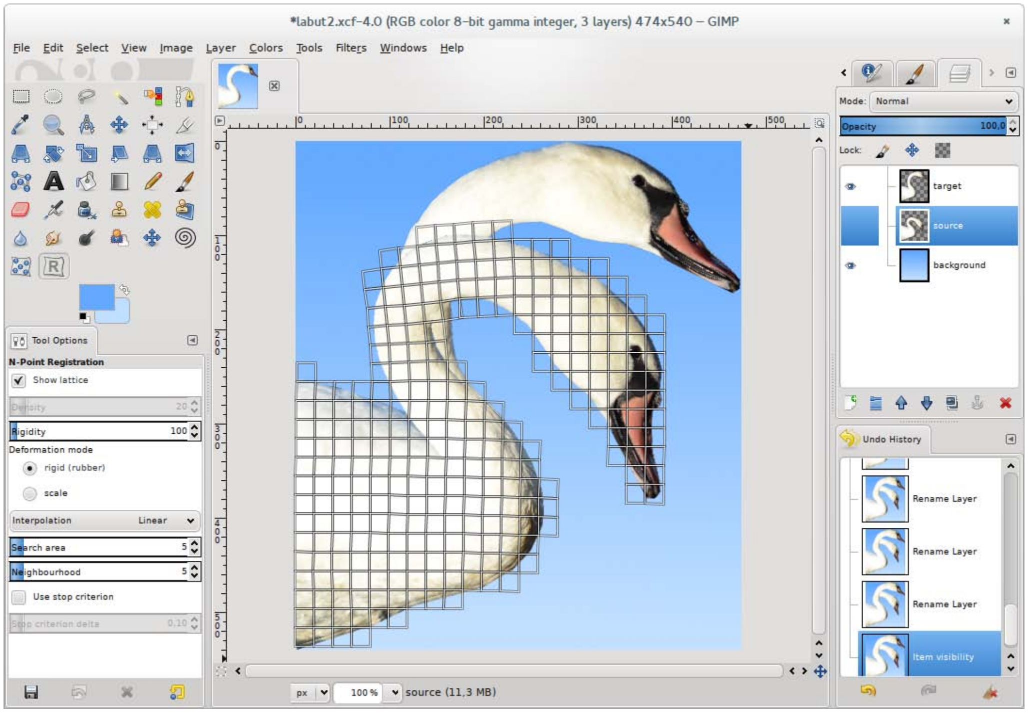
Description of user interface

Figure 3.3 depicts how the n-point image deformation tool is integrated into GIMP's GUI.

After activation of the tool, it is necessary to specify the source and the target image which enters the registration process. Every image in GIMP can be composed of several layers. The target image is always the first layer and the source image is the active layer. In Figure 3.3, layers are depicted at the upper right corner. The source image is the picture of a swan over which the lattice is constructed, the target image is the picture of a swan with neck further away from its body.

The registration process is started after clicking on the canvas. After that, a lattice appears over the source image and user can manipulate the image in the same way as with the n-point deformation tool.

The behaviour of the tool can be adjusted using the following set of settings (we will describe only the settings that differ from the deformation tool settings):


*labut2.xcf-4.0 (RGB color 8-bit gamma integer, 3 layers) 474x540 – GIMP

N-Point Registration
Show lattice [✓]
Density 20
Rigidity 100
Deformation mode: [✓ rigid (rubber)] scale
Interpolation Linear
Search area 5
Neighbourhood 5
Use stop criterion [ ]
Stop criterion delta 0,10

Mode: Normal
Opacity 100,0

Figure 3.3: *N-point image registration tool in development version (2.9) of GIMP*

- An option to specify the values of block-matching parameters described on page 30, that is “search area” (M) and “neighbourhood” (N).
- An option to use the criterion for automatically stopping the registration process according to d_{avg} formula described on page 31.
  - Registration is paused after 20 iterations during which values of d_{avg} differ by less than the specified delta from the average value $\bar{d}_{\text{avg}}$ within these 20 iterations.
  - When the registration is paused, user cannot manipulate with the deformed image. Registration is resumed when this option is unchecked or when the value of some the other options is changed (e.g. M or N).

If user wants to apply the deformation, the tool can be stopped by pressing ENTER, he can cancel the registration by pressing ESC.

Current problems of the tool

The n-point registration tool inherits from its parent both benefits and drawbacks. Again, there is the problem with large images. In this tool, the problem also is caused by block-matching method that is employed during registration. For large images, it is necessary to set a higher value of the search parameters ("search area" and "neighbourhood"). A possible solution is to use the hierarchical approach (see section 2.3.3) where we start registering on scaled down image and gradually refine on larger images.

The tool currently does not use multiple CPU or GPU cores to speed up computations of the block-matching.

Chapter 4

Experiments

The n-point image registration (NPR) tool was tested on a set of images of various type. Results of registration (i.e. deformed images) using the tool were compared with results of registration using Drop [6] and NiftyReg [21] tools.

This chapter briefly describes the mentioned methods and tools and the results of experiments are presented here as well.

4.1 Drop

Drop is a tool allowing to perform non-linear image registration as well as 3D registration. It is based on a method described in [6]. The method is an area-based method that uses free-form deformation model for deforming the source image. The model is given by n control points which form a grid. These points are fitted to a mapping function which can be linearly interpolated but also interpolated e.g. using cubic B-splines [25]. Thanks to employed optimization strategy, one can select an arbitrary (dis)similarity measure [6]. The tool allows us to choose from a large number of these measures – for example SAD, SSD, NCC, NMI (normalized mutual information). Energetic function which is expressed using MRF (Markov Random Fields) and which consists of similarity measure term and regularization term is optimized employing discrete optimization (FastPD algorithm) [6].

The tool contains GUI that is comfortable to use. When needed, the registration can be run also on command prompt.

The method uses hierarchical approach. The tool allows us to set a number of levels of image resolution (in the results section below we will refer to them as `imagelevels`) and also a number of levels of grid squares (or rectangles) sizes (`gridlevels`). We start with an initial size of grid squares sizes (`gridsize`) or in other words with grid of control points with spacing specified by `gridsize`. With every subsequent level the grid is refined, i.e. spacings are reduced by a half. In the results section below, the number of iterations performed in every level will be called `iterations` and a similarity measure will be called `measure`. The influence of the regularization term can be adjusted using parameter λ.

The tool has been created for medical usage but it can be also used for other purposes – for example, in [28] it is used to improve the final result of ARAP registration of hand-drawn images.

Drop is able to load images of various medical formats – e.g. MHD (MetaImage), Analyze, DICOM – but can also load classic raster PNG and BMP images. The loaded images are converted to shades of gray, that means that the color information is not utilized during registration. The result of registration can be saved to a file in MHD format only. In a situation when color images enter the registration process and we would like to obtain color resulting image, the tool allows to export a deformation (displacement) field. The field is exported into two MHD format files. Both files contain for every pixel of the source image one *float* number that represents a displacement along x axis (one file) and y axis (the other file). The deformed color image can be obtained by applying this deformation field on the source image (the field represents backward transformation).

The tool is available at `http://www.mrf-registration.net` where a user manual describing various setting possibilities of the tool is also available.

4.2 NiftyReg

NiftyReg is a tool allowing to perform rigid or affine image registration as well as non-linear image registration. It can also perform 3D registration. Implementation of non-linear registration is based on method decribed in [21] and [25].

The method is area-based method which uses free-form deformation model based on cubic B-splines for deforming the source image. Again, the model is specified by n control points that form a grid. As a (dis)similarity measure, we can select one of SSD, NMI or KLD (Kullback–Leibler divergence). Energetic function which consists of a similarity measure term and a regularization term is optimized employing conjugate gradient method (gradient ascent) [21].

Unlike the Drop tool, NiftyReg does not contain GUI. It can be run on command prompty only.

The method uses hierarchical approach. The number of levels can be specified by parameter `ln`. Let us consider that we have a square grid of control points. We start with a grid of control points with spacing that equals $\texttt{sx}\cdot 2^{\texttt{ln}-1}$ and we finish with a grid with spacing given by optional parameter `sx`. The maximum number of optimizing iterations that is performed in one level can be specified by parameter `maxit`. We can also specify weights of (sub)terms the regularization term consists of. These weights penalize some unwanted properties of the sought deformation. Weight of a term penalizing bending [25] can be specified by parameter `be`. In the results section below, a similarity measure will be called `measure`.

This tool is able to load images of medical format NII (NIfTI) as well as e.g. images of raster graphics format PNG. All image output produced by the tool is in NII format only. As in the case of software Drop, even NiftyReg converts color input images to grayscale colors. Again, color output can be obtained from deformation field which can be exported into a single file. One half of content of the file represents a deformation along x axis, the other half deformation along y axis.

The tool is available at `http://sourceforge.net/projects/niftyreg/`.

4.3 Results

The results of image registration using n-point registration (NPR) tool, using Drop and NiftyReg are presented on the following pages. To every single resulting image there is a set of parameters θ describing tool's settings written in tables. Parameters of NPR tool are described on page 46 and 48. Parameters of Drop and NiftyReg were described above.

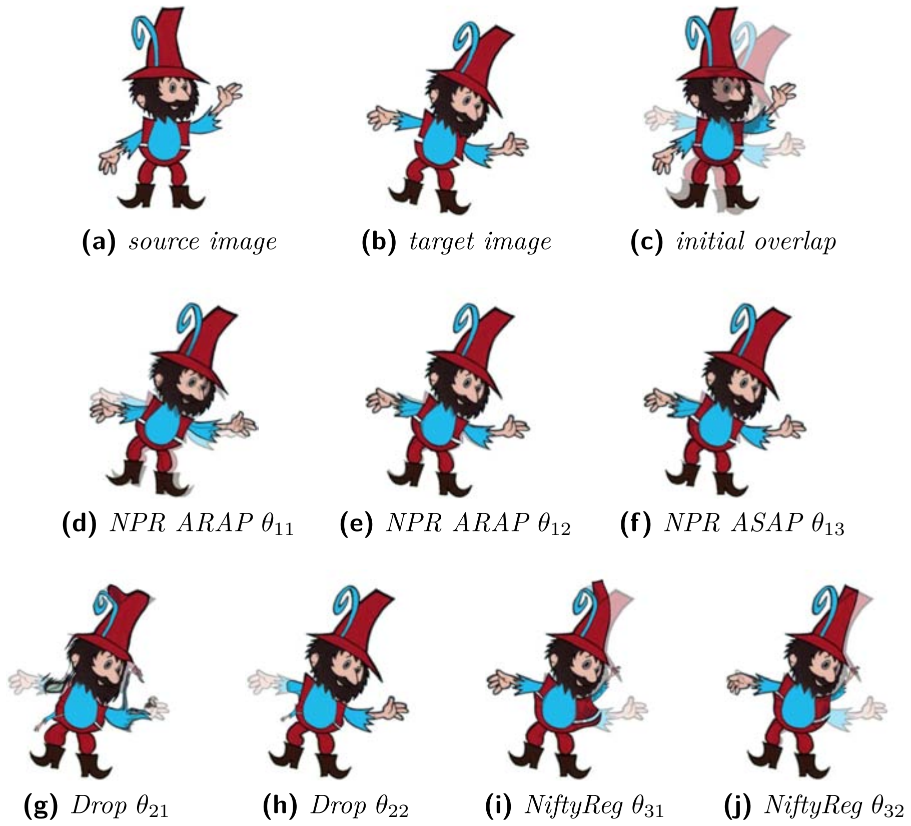
The following three-phase procedure was chosen for experiments using the NPR tool. The parameters used in these phases are indicated by θ_{11}, θ_{12}, θ_{13}. In the first phase the registration was usually started with high rigidity value. By that the source image quickly roughly aligned with the target image. In the second phase and with registration still running, the rigidity was decreased. By that we achieved more accurate aligning of the source image with the target image. In the last phase, deformation model was changed from ARAP to ASAP which allowed a local change of scale. By that the process was finished.

Although NPR tool allows interactive registration (see section 2.5.3 on page 32) it was not used during experiments.

When experiments using Drop and NiftyReg took place, for every choice of parameters, the registration was started from the beginning.

The parameters θ_{12} (or θ_{13}), θ_{22} a θ_{32} describe settings giving the best results that I was able to achieve within experiments.

Images presented on the following pages are available on the attached media in full resolution.


(a) *source image* **(b)** *target image* **(c)** *initial overlap*

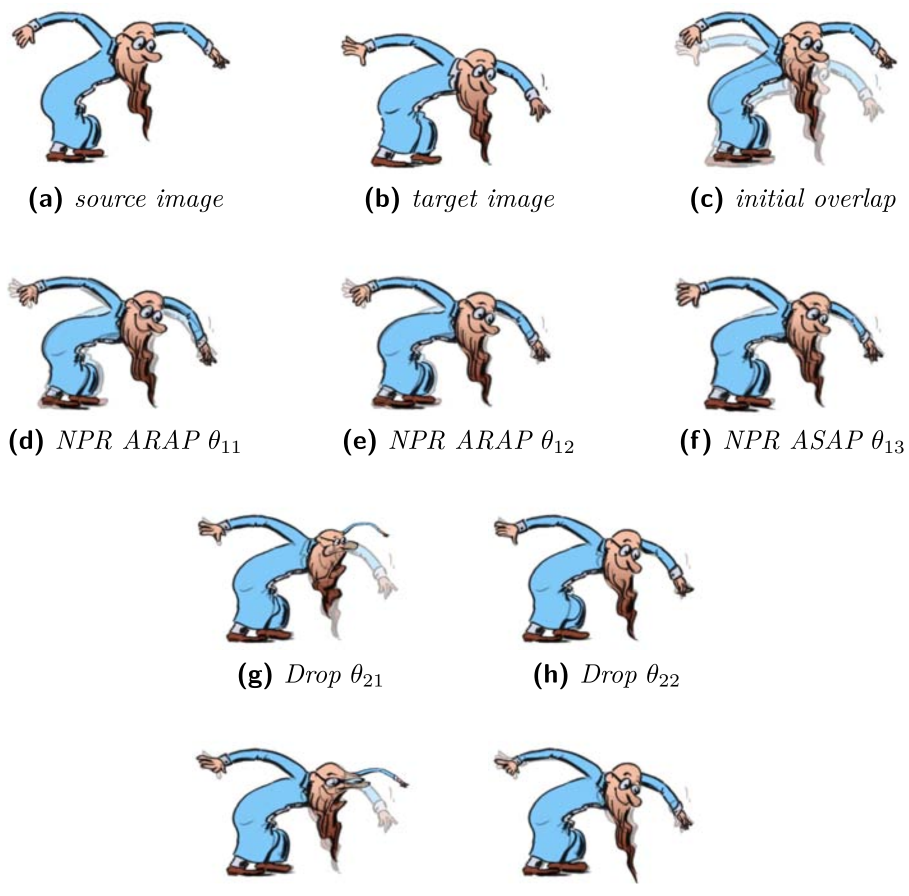
(d) *NPR ARAP* θ_{11} **(e)** *NPR ARAP* θ_{12} **(f)** *NPR ASAP* θ_{13}

(g) *Drop* θ_{21} **(h)** *Drop* θ_{22} **(i)** *NiftyReg* θ_{31} **(j)** *NiftyReg* θ_{32}

Figure 4.1: *Experiment 1 – methods results; images resolution:* 420×541

Tool	Image	Parameters	
NPR	**(d)**	$\theta_{11} =$	$\{$density = 16, rigidity = 600, mode = rigid, $M = 24$, $N = 8\}$
NPR	**(e)**	$\theta_{12} =$	$\{$density = 16, rigidity = 30, mode = rigid, $M = 20$, $N = 8\}$
NPR	**(f)**	$\theta_{13} =$	$\{$density = 16, rigidity = 30, mode = scale, $M = 20$, $N = 8\}$
Drop	**(g)**	$\theta_{21} =$	$\{$gridsize = 64, imagelevels = 5, gridlevels = 5, iterations = 5, measure = SAD, $\lambda = 10\}$
Drop	**(h)**	$\theta_{22} =$	$\{$gridsize = 128, imagelevels = 5, gridlevels = 5, iterations = 5, measure = SAD, $\lambda = 10\}$
NiftyReg	**(i)**	$\theta_{31} =$	$\{$sx = 25, ln = 5, maxit = 300, measure = SSD, be = 0$\}$
NiftyReg	**(j)**	$\theta_{32} =$	$\{$sx = 29, ln = 5, maxit = 350, measure = SSD, be = 0$\}$

Table 4.1: *Experiment 1 – methods parameters*



(a) *source image* **(b)** *target image* **(c)** *initial overlap*

(d) *NPR ARAP* θ_{11} **(e)** *NPR ARAP* θ_{12} **(f)** *NPR ASAP* θ_{13}

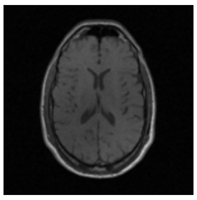
(g) *Drop* θ_{21} **(h)** *Drop* θ_{22}

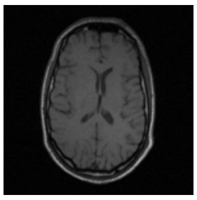
(i) *NiftyReg* θ_{31} **(j)** *NiftyReg* θ_{32}

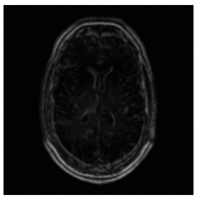
Figure 4.2: *Experiment 2 – methods results; images resolution:* 560×400

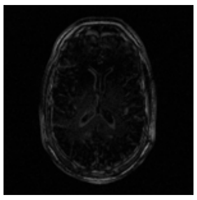
Tool	**Image**	**Parameters**	
NPR	**(d)**	$\theta_{11} =$	{density = 16, rigidity = 200, mode = rigid, $M = 20$, $N = 8$}
NPR	**(e)**	$\theta_{12} =$	{density = 16, rigidity = 30, mode = rigid, $M = 20$, $N = 8$}
NPR	**(f)**	$\theta_{13} =$	{density = 16, rigidity = 30, mode = scale, $M = 20$, $N = 8$}
Drop	**(g)**	$\theta_{21} =$	{gridsize = 64, imagelevels = 3, gridlevels = 3, iterations = 5, measure = SAD, $\lambda = 10$}
Drop	**(h)**	$\theta_{22} =$	{gridsize = 128, imagelevels = 4, gridlevels = 4, iterations = 5, measure = SAD, $\lambda = 10$}
NiftyReg	**(i)**	$\theta_{31} =$	{sx = 25, ln = 5, maxit = 100, measure = SSD, be = 0}
NiftyReg	**(j)**	$\theta_{32} =$	{sx = 29, ln = 5, maxit = 300, measure = SSD, be = 0}

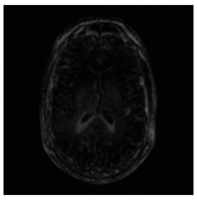
Table 4.2: *Experiment 2 – methods parameters*

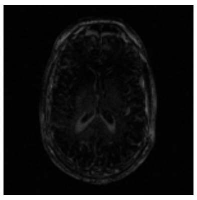

(a) *source image*

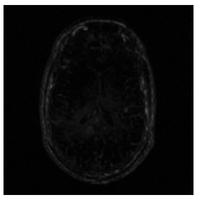

(b) *target image*

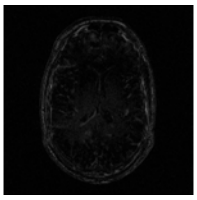

(c) *initial difference*

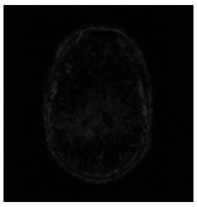

(d) *NPR ARAP* θ_{11}


(e) *NPR ARAP* θ_{12}


(f) *NPR ASAP* θ_{13}


(g) *Drop* θ_{21}


(h) *Drop* θ_{22}


(i) *NiftyReg* θ_{31}

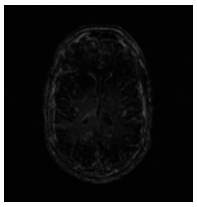
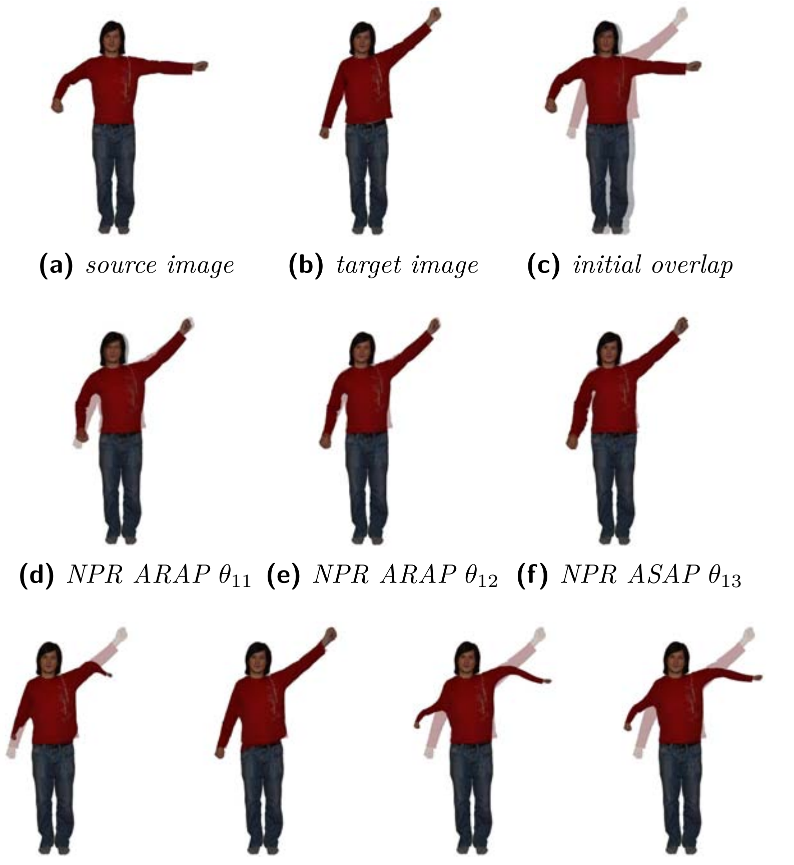

(j) *NiftyReg* θ_{32}

Figure 4.3: *Experiment 3 – methods results; images resolution:* 421×436*. To show the resulting overlap, difference images have been used. In terms of similarity, Figures* (**g**) *a* (**i**) *are closer to the target image than Figures* (**h**) *and* (**j**)*. However, the deformation is so large that the original structure of the brain disappears. Therefore, as a better result, images in Figures* (**h**) *and* (**j**) *have been selected.*

Tool	**Image**	**Parameters**	
NPR	**(d)**	$\theta_{11} =$	{density = 10, rigidity = 50, mode = rigid, $M = 5$, $N = 10$}
NPR	**(e)**	$\theta_{12} =$	{density = 10, rigidity = 5, mode = rigid, $M = 5$, $N = 15$}
NPR	**(f)**	$\theta_{13} =$	{density = 10, rigidity = 10, mode = scale, $M = 5$, $N = 15$}
Drop	**(g)**	$\theta_{21} =$	{gridsize = 32, imagelevels = 3, gridlevels = 3, iterations = 5, measure = SAD, $\lambda = 10$}
Drop	**(h)**	$\theta_{22} =$	{gridsize = 32, imagelevels = 3, gridlevels = 3, iterations = 5, measure = SAD, $\lambda = 100$}
NiftyReg	**(i)**	$\theta_{31} =$	{sx = 5, ln = 5, maxit = 300, measure = SSD, be = 0}
NiftyReg	**(j)**	$\theta_{32} =$	{sx = 5, ln = 5, maxit = 300, measure = SSD, be = 0.005}

Table 4.3: *Experiment 3 – methods parameters*



(a) *source image* **(b)** *target image* **(c)** *initial overlap*

(d) *NPR ARAP* θ_{11} **(e)** *NPR ARAP* θ_{12} **(f)** *NPR ASAP* θ_{13}

(g) *Drop* θ_{21} **(h)** *Drop* θ_{22} **(i)** *NiftyReg* θ_{31} **(j)** *NiftyReg* θ_{32}

Figure 4.4: *Experiment 4 – methods results; images resolution:* 489×656

Tool	Image	Parameters
NPR	**(d)**	$\theta_{11} =$ {density = 16, rigidity = 200, mode = rigid, $M = 24$, $N = 8$}
NPR	**(e)**	$\theta_{12} =$ {density = 16, rigidity = 30, mode = rigid, $M = 10$, $N = 15$}
NPR	**(f)**	$\theta_{13} =$ {density = 16, rigidity = 30, mode = scale, $M = 10$, $N = 15$}
Drop	**(g)**	$\theta_{21} =$ {gridsize = 128, imagelevels = 4, gridlevels = 4, iterations = 5, measure = SAD, $\lambda = 10$}
Drop	**(h)**	$\theta_{22} =$ {gridsize = 192, imagelevels = 7, gridlevels = 5, iterations = 5, measure = SAD, $\lambda = 10$}
NiftyReg	**(i)**	$\theta_{31} =$ {sx = 305, ln = 5, maxit = 150, measure = SSD, be = 0}
NiftyReg	**(j)**	$\theta_{32} =$ {sx = 35, ln = 5, maxit = 100, measure = SSD, be = 0}

Table 4.4: *Experiment 4 – methods parameters*

Conclusion

In this thesis, I utilized the experience gained through summer 2013 and also other experiences gained during studying. Into the development version of GIMP I have implemented a tool allowing image registration preserving rigidity.

Results showing the functionality of the tool have been presented in the chapter with experiments in which the tool was compared with some other image registration methods. From these experiments it is evident that some modern non-linear image registration methods, when properly set, are able to cope even with a large deformation of images entering the registration. In contrast to these methods a great advantage of ARAP registration method can, in some situations, be the fact, that it produces, in almost all circumstances, the results that are not unnaturally deformed. This feature of the method is particularly useful when registering real or cartoon figures and their poses.

The process of integrating a new, wider, functionality into well-established program as GIMP, is not of the easiest – it is composed of many subproblems and takes a lot of time. The tool, which was developed within this thesis is functional however still not perfect – some of its weaknesses has been described in the chapter on implementation. Further work is thus, among other things, to eliminate those weaknesses.

Thanks to this thesis, I broaden my horizons some more. I am grateful for the oportunity to be involved in interesting projects connected to this thesis.

Bibliography

[1] Alexa, M.; Cohen-Or, D.; Levin, D.: As-rigid-as-possible shape interpolation. In *ACM SIGGRAPH Conference Proceedings*, 2000, pp. 157–164.

[2] Botsch, M.; Pauly, M.; Wicke, M.; etc.: Adaptive Space Deformations Based on Rigid Cells. *Computer Graphics Forum*, volume 26, no. 3, 2007: pp. 339–347.

[3] Brown, L. G.: A Survey of Image Registration Techniques. *ACM Computing Surveys*, volume 24, no. 4, 1992: pp. 325–376.

[4] Bueno, J.: Add your own GIMP features – Dive into the code base of the GNU Image Manipulation Program. *IBM developerWorks*, 2010.

[5] Glasbey, C. A.; Mardia, K. V.: A review of image-warping methods. *Journal of Applied Statistics*, volume 25, no. 2, 1998: pp. 155–171.

[6] Glocker, B.; Komodakis, N.; Tziritas, G.; etc.: Dense Image Registration through MRFs and Efficient Linear Programming. 2008.

[7] Golub, G.; Van Loan, C.: *Matrix Computations.* Johns Hopkins Studies in the Mathematical Sciences, 1996, ISBN 9780801854149.

[8] Goshtasby, A.: Image Registration Methods. In *Image Registration*, Advances in Computer Vision and Pattern Recognition, Springer London, 2012, pp. 415–434.

[9] Goshtasby, A.: Robust Parameter Estimation. In *Image Registration*, Advances in Computer Vision and Pattern Recognition, Springer London, 2012, pp. 313–341.

[10] Goshtasby, A.: Similarity and Dissimilarity Measures. In *Image Registration*, Advances in Computer Vision and Pattern Recognition, 2012, pp. 7–66.

[11] Goshtasby, A.: Transformation Functions. In *Image Registration*, Advances in Computer Vision and Pattern Recognition, Springer London, 2012, pp. 343–400.

[12] Heckbert, P.: *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, University of California, Berkeley, 1989.

[13] Higham, N. J.; Robert; Schreiber, S.: Fast polar decomposition of an arbitrary matrix. *SIAM Journal on Scientific and Statistical Computing*, volume 11, 1990: pp. 648–655.

[14] Igarashi, T.; Moscovich, T.; Hughes, J. F.: As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, volume 24, no. 3, 2005: pp. 1134–1141.

[15] Kavan, L.; Collins, S.; Žára, J.; etc.: Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, 2007, pp. 39–46.

[16] Lewis, J. P.; Cordner, M.; Fong, N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *ACM SIGGRAPH Conference Proceedings*, 2000, pp. 165–172.

[17] Li, W.; Salari, E.: Successive Elimination Algorithm for Motion Estimation. *IEEE Transactions on Image Processing*, volume 4, no. 1, 1995: pp. 105–107.

[18] Lowe, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, volume 60, no. 2, 2004: pp. 91–110.

[19] Lucas, B. D.; Kanade, T.: An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, volume 2, 1981, pp. 674–679.

[20] Martin, J.; Crowley, J. L.: Experimental Comparison of Correlation Techniques. In *IAS-4, International Conference on Intelligent Autonomous Systems*, 1995.

[21] Modat, M.; Ridgway, G. R.; Taylor, Z. A.; etc.: Fast Free-form Deformation Using Graphics Processing Units. *Computer Methods and Programs in Biomedicine*, volume 98, no. 3, 2010: pp. 278–284.

[22] Ponce, J.; Karahoca, A.: *Data Mining and Knowledge Discovery in Real Life Applications*. In-Teh, 2009, ISBN 9783902613530.

[23] Porter, T.; Duff, T.: Compositing Digital Images. *SIGGRAPH Computer Graphics*, volume 18, no. 3, 1984.

[24] Rivlin, T.: Least Squares Approximation. In *An Introduction to the Approximation of Functions*, Dover Publications, 1981.

[25] Rueckert, D.; Sonoda, L. I.; Hayes, C.; etc.: Nonrigid registration using free-form deformations: application to breast MR images. *IEEE Transactions on Medical Imaging*, volume 18, no. 8, 1999: pp. 712–721.

[26] Schaefer, S.; McPhail, T.; Warren, J.: Image deformation using moving least squares. *ACM Transactions on Graphics*, volume 25, no. 3, 2006: pp. 533–540.

[27] Sorkine, O.; Alexa, M.: As-rigid-as-possible Surface Modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, 2007, pp. 109–116.

[28] Sýkora, D.; Dingliana, J.; Collins, S.: As-rigid-as-possible Image Registration for Hand-drawn Cartoon Animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, 2009, pp. 25–33.

[29] Wang, Y.; Xu, K.; Xiong, Y.; etc.: 2D Shape Deformation Based on Rigid Square Matching. *Computer Animation and Virtual Worlds*, volume 19, no. 3–4, 2008: pp. 411–420.

[30] Winter, S.; Brendel, B.; Pechlivanis, I.; etc.: Registration of CT and Intraoperative 3-D Ultrasound Images of the Spine Using Evolutionary and Gradient-Based Methods. *IEEE Transactions on Evolutionary Computation*, volume 12, no. 3, 2008: pp. 284–296.

[31] Zitová, B.; Flusser, J.: Image registration methods: a survey. *Image and Vision Computing*, volume 21, 2003: pp. 977–1000.

Appendix A

Acronyms

ARAP	As-Rigid-As-Possible
ASAP	As-Similar-As-Possible
FLOSS	Free/Libre/Open Source Software
GEGL	Generic Graphics Library
GIGO	Garbage In, Garbage Out
GIMP	GNU Image Manipulation Program
GPL	General Public Licence
GSOC	Google Summer of Code
GUI	Graphical User Interface
MLS	Moving Least Squares
MoCap	Motion Capture
NCC	Normalized Cross-Correlation
NPR	N-Point Registration
SAD	Sum of Absolute Differences
SIMD	Single Instruction Multiple Data
SSD	Sum of Squared Differences
SVD	Singular Value Decomposition

Appendix B

Contents of enclosed CD

```
│
├── readme.txt ............................ brief description of the CD
├── bin ......... directory with the executable form of the implementation
├── results .................................... results of experiments
├── src
│   ├── impl ........................ source code of the implementation
│   └── thesis ....................... source text of the thesis in LaTeX
├── text
│   ├── thesis-en.pdf ........ english version of the thesis in PDF format
│   └── thesis-cz.pdf .......... czech version of the thesis in PDF format
└── video ............. video clips presenting the capabilities of the tools
```