FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Correction of Camera Registration Errors through Optical Flow

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Christian Hafner
Matrikelnummer 0925172

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Projektass. Dipl.-Ing. Murat Arikan

Wien, 03.09.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Correction of Camera Registration Errors through Optical Flow

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Christian Hafner
Registration Number 0925172

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass. Dipl.-Ing. Murat Arikan

Vienna, 03.09.2013

(Signature of Author) (Signature of Advisor)

Abstract

In order to digitize artifacts of the real world, laser scanners with mounted cameras are being used. Reliable strategies to capture the geometry are known, but the mapping of color information onto the geometry still forms an obstacle. One reason for this is the misalignment between the scanner and the camera, which results in an inaccurate reprojection of the images onto the acquired mesh. Since different photographs are used to color different sections of the model, discontinuities will be visible around the seams.

This thesis aims to provide a method to correct these discontinuities in an automated fashion. The optical flow between two neighboring photographs is calculated and used to warp the image space at the seams. The resulting algorithm is independent of geometric complexity and deals well with heterogeneous lighting conditions.

Contents

1 Introduction **1**

2 Related Work **5**
2.1 Labeling 5
2.2 Stitching 6

3 Optical Flow **9**
3.1 Brute Force Method 10
3.2 Hierarchical Method 12
3.3 Floating Textures 14

4 Stitching Algorithm **17**

5 Results **23**

6 Conclusion **27**

7 Listings **29**
7.1 Brute force Optical flow shader 29
7.2 Stitching shader 31

Bibliography **33**

CHAPTER 1

Introduction

Digitization of excavation sites, architectural structures and artifacts has become an important goal to cultural heritage preservation. To record the shape of an object, professional laser measurement systems are employed. These scan all surfaces and create high-resolution point clouds. To record the color of an object at each point, a camera shoots photographs from different angles. In order to project a photograph onto the recorded geometry, the camera's position and orientation have to be registered in the coordinate system of the laser scanner. Additionally, the intrinsic parameters of the camera are used to compensate for the radial and tangential distortion of the photograph. In theory, this should enable us to map every photo to the geometry accurately. In practice the mapping will be faulty for a variety of reasons:

- The recorded position and orientation of the camera in the scanner's coordinate system will be slightly defective. This happens because of registration errors that cannot be avoided with currently available equipment. When the photograph is back-projected onto the geometry, it will not be aligned perfectly. If the same section of a mesh is depicted on two photographs, their projections will also not match up and create ghosting artifacts.
- Scanners are often employed in environments where lighting conditions cannot be controlled very well. Therefore data sets can contain photographs that have been illuminated under different lighting setups, resulting in highlights and varying degrees of brightness.
- Dense point clouds are simplified to afford real-time rendering. Details may be lost during triangulation and the photographs might not perfectly conform to the geometry anymore, especially around silhouettes of objects. As a consequence of this, the color information of objects will be projected onto the background or vice versa.

Photographic data sets usually contain a high degree of redundancy which means that one section of the model is visible in many photographs. This leads to the question of choosing an appropriate integration strategy. Essentially there are two possibilities:

1. **Partition-based:** For each triangle in a mesh, we choose one photograph to color it. This results in a labeling
$$\{V_1, \dots, V_n\} \to \{P_1, \dots, P_k\},$$
where $\{V_1, \dots, V_n\}$ denotes the set of triangles and $\{P_1, \dots, P_k\}$ denotes the set of photographs. The set of all triangles that are mapped to P_i is called the label of P_i. The main disadvantage of this method is the existence of seams. If two neighboring triangles are colored by different photographs, there will be a visible discontinuity in between due to the reasons mentioned above. This effect is illustrated in Figure 1.1.

2. **Blending-based:** Each triangle can be colored by many photographs by combining their color values to form a final value. This results in a map
$$\{V_1, \dots, V_n\} \to \wp(\{P_1, \dots, P_k\}).$$
By choosing weights for all contributing photos, seams can be avoided. Misaligned photos will typically create ghosting artifacts instead (see Figure 1.2).

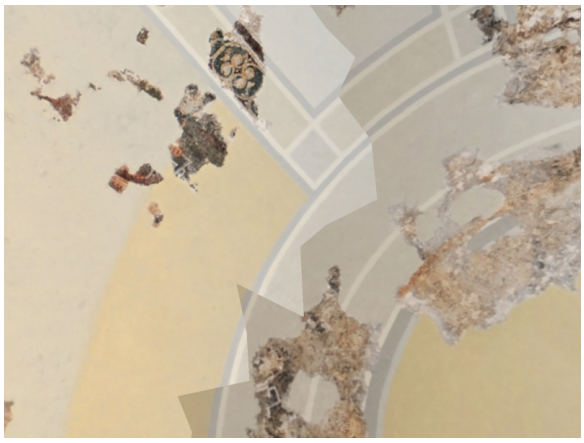

Figure 1.1: Example of discontinuity artifacts that appear on the edge between two labels with misaligned photos. The white and gray lines in the painting do not match up at the seams. The right photo is also visibly darker than the left, making the seam even more distinctive.

This work assumes that a partition-based strategy is used and provides a method to avoid discontinuities at the seams. Since a global alignment of all photographs is impossible to achieve in practice, we will attempt to optimize it locally. We follow the approach described in [Dellepiane et al., 2012], who achieve a blending-based image-to-geometry registration. The main contribution of this thesis is to adapt and implement their algorithm in order to form a partition-based algorithm.

The method makes use of information redundancy around the seam, i.e. the area around the seam is visible on both of the neighboring photographs. By comparing these two photographs we can estimate the alignment errors introduced by camera misregistration and correct them with

Figure 1.2: Example of ghosting artifacts that appear when misaligned photos are combined using blending. Image taken from [Dellepiane et al., 2012].

local warp fields. These warp fields consist of displacement vectors for each pixel of the two images and can be found with optical flow methods. We explain and evaluate different methods for calculating these warp fields in section 3.

Once the warp fields for the areas around all seams between neighboring labels have been found, they can be applied to produce a warped texture for each label that minimizes discontinuity artifacts. For each warp field that affects a label, we create a weight field to smooth the effect of the warping. Directly at the seams, each of the two photographs will be warped to compensate for 50% of the misalignment. The farther a pixel is away from the seam, the shorter the displacement vector will be. The algorithm is detailed in section 4.

CHAPTER 2

Related Work

2.1 Labeling

After data acquisition, one is faced with an extensive amount of geometric and photographic data, the latter of which contains a high degree of redundancy. Strategies to make efficient use of this have been proposed in previous works.

Callieri et al. introduce a quality metric to create a grayscale mask for each photograph in [Callieri et al., 2008]. To create these masks they use a combination of different metrics. The authors propose penalizing steep viewing angles, high distances between camera and surface, and silhouettes, since these sections of a photograph suffer from loss of detail. The values of all metrics are multiplied to preserve zeroes. The final color value of a pixel is calculated by blending the photographs with weights according to the masks.
As a result of this method, discontinuities are avoided due to the smooth nature of the image masks. Although only high-quality sections of each photograph are chosen, the blending of misaligned images leads to ghosting artifacts and blurry results.

[Dellepiane et al., 2012] uses this quality metric to create a labeling partition of the mesh, where each surface point is assigned to the photograph with the highest quality value. One problem of this method is the introduction of a great many of seams. Irregularities in the surface of a mesh can lead to high-frequency variations in the mask produced by the angle metric. The resulting partition possesses speckled regions where cells do not consist of a small amount of connected components anymore. Instead they contain many loose faces and therefore many seams.

A different approach is proposed by Lempitsky and Ivanov in [Lempitsky and Ivanov, 2007]. The authors represent the question for a suitable labeling as a Markov Random Field energy minimization problem. Each possible labeling can be assigned an energy term, which is sought to be minimized. The energy value is defined as the sum of two terms:

1. The data cost, which is the sum of the weights calculated by the quality metric for every face.

2. The smoothness cost, which penalizes adjacent faces with large color distances along their shared edge.

This algorithm produces larger homogeneous patches and fewer seams.

To fight seam discontinuities, Gal et al. [Gal et al., 2010] propose a method that uses MRF labeling. They observe that errors resulting from camera misregistrations are usually translational in nature. To correct these, tuples $(P_i, \mathbf{s})$ are used as labels. P_i is a photograph and $\mathbf{s}$ is a shift vector in $\mathbb{R}^2$. Since the energy term in [Lempitsky and Ivanov, 2007] penalizes large color distances along edges, the shifted photographs provide more room for minimization. A coarse-to-fine strategy is used to limit the number of resulting labels.

MRF labeling with and without shift vectors is explored by Birsak in [Birsak, 2012]. It is shown that the labeling strategy with shift vectors produces undesirable results if the misregistration errors are overestimated. Moreover the computation for a complex model with 70 photographs can take up to many days.

2.2 Stitching

This section explores previous attempts to correct ghosting and discontinuity artifacts that result from camera registration errors or varying lighting conditions. Two fundamental approaches can be identified: finding a global registration of all images or using local optimization methods.

A global optimization strategy is proposed in [Lensch et al., 2000]. The algorithm finds a view point for each image that minimizes the distance between the silhouette of the geometry and the silhouette extracted from the image. A variant of the simplex algorithm with multiple starting points is used to find a global minimum. In a second phase, the color information is used to align the features between different images. Successively, the algorithm selects each image and calculates the total color distance to all other images in overlapping areas. The simplex method is used find an optimized view point that minimizes this term. This process is iterated until the results do not change anymore. This method produces accurate results in controlled environments but relies on several factors, like the accurateness of the 3D model and perfectly perspective images. These may not always be given in real world applications.

To deal with imperfect 3D models and registration errors, Takai et al. provide a local optimization strategy in [T. Takai and Matsuyama, 2010]. Textures are locally transformed around the vertices of a simplified version of the 3D model. This is achieved by finding an image feature close to the projection of a vertex and tracking this feature in all contributing images. By adjusting the texture coordinates appropriately, the textures are warped such that the image feature aligns in all images. The method described is view dependent and cannot be used to find a globally valid image registration.

The paper's method is based on the work of Dellepiane et al. [Dellepiane et al., 2012]. Their approach uses the quality metric from [Callieri et al., 2008] in combination with an optical flow strategy. The basic algorithm works as follows:

1. Calculate the face weights for each label according to [Callieri et al., 2008].

2. Derive a "two best fragments" labeling from the weights, such that each triangle is mapped onto the pair of images with the two highest weights. If a label L_{IJ} consists of the images I and J, I is said to be the "dominant" image space for that region of the mesh.

3. For each label L_{IJ}: find all other photographs that show a significantly large part of L_{IJ} and transform them into L_{IJ}'s dominant image space.

4. Between each transformed image and the photograph L_{IJ}, calculate the optical flow to create a warp field. This warp field will align the features of the two images and counter any misregistration error.

5. The final color value for a fragment in label L_{IJ} is calculated as the weighted sum of the photograph L_{IJ} and all other previously transformed labels. To ensure a smooth transition between two neighboring labels with different dominant image spaces L_{IJ} and L_{JI}, the warp fields themselves are interpolated in the area around the seam.

By using color blending in combination with optical flow warp fields, ghosting artifacts and brightness discontinuities can be avoided. Furthermore, the interpolation between warp fields counters feature discontinuities.

This method suffers most from its labeling approach. As noted above, a "best fragment" labeling already yields inhomogeneous results. A "two best fragments" labeling further subdivides every label and creates an even more disconnected set.

CHAPTER 3

Optical Flow

Optical flow attempts to describe the motion of objects with respect to the observer between a sequence of images. An object is thought of as a set of surface points. These can move over time, either when the object itself moves or when the observer moves. Thus each surface point follows a path, and the projection of the point onto the image plane follows a path as well. Every pixel of an image corresponds to a surface point and can be mapped onto its instantaneous velocity in image space. This mapping is known as the 2D motion field and optical flow methods aim to estimate it.

As a starting point, assumptions about the behavior of the observed objects have to be made. One common constraint is the *brightness constancy assumption*. It says that one surface point exhibits the same brightness during two consecutive frames. This means that the point will produce a pixel of the same brightness on both frames:

$$I(x,y,t) = I(x+u, y+v, t+\Delta t)$$

I signifies the image intensity function, (x,y) is the pixel on the first frame, (u,v) is the 2D motion vector, t is the time of the first photo and Δt is a small timestep.

For estimating the optical flow between photographs of a large data set, whose images have been taken with long time intervals in between and under different lighting conditions, the brightness constancy assumption does not hold in this form. For the optical flow implementations that are used in this paper, we modify the assumption.
Overall brightness varies from photograph to photograph. But it can be observed that small sections in one photograph rarely suffer from high lighting variances. If the same section of an object is depicted on two photographs under different lightning conditions, the difference between a pixel's intensity and the averaged intensity of its surrounding area turns out to be more robust than the absolute brightness of a pixel. An illustration can be seen in Figure 3.1.

Another constraint that suits our data sets is the *smoothness constraint*. It is derived from the fact that we deal with photographs of rigid bodies. Two neighboring surface points on a rigid body that is translated, rotated or scaled, move in a similar fashion. Therefore the motion

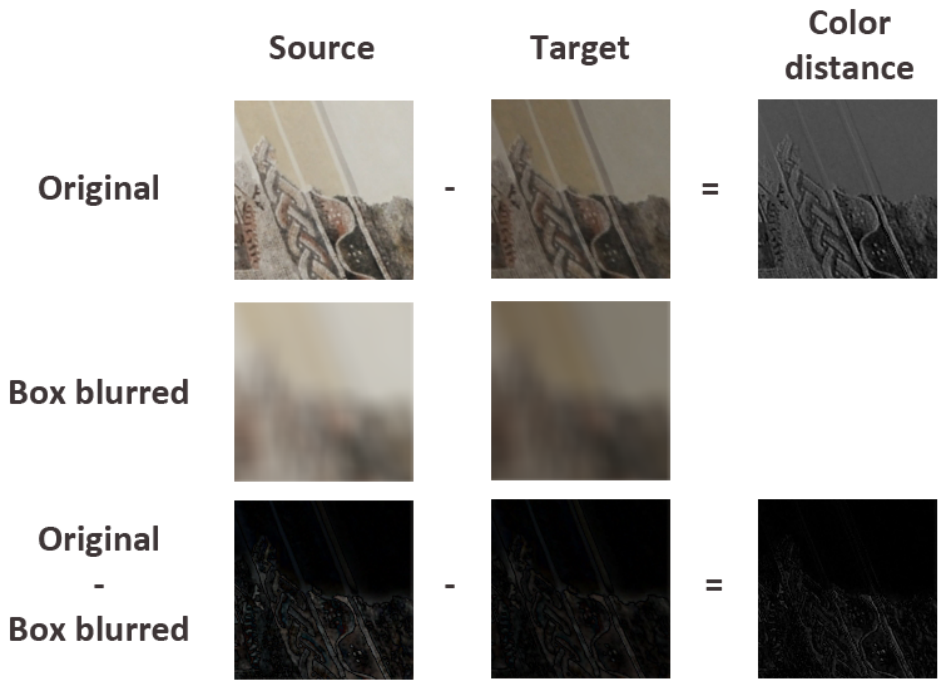

Figure 3.1: The first row shows the color distance between the absolute pixel values of source and target image. The second row shows a convolution of source and target image with a box filter kernel. The third row shows the difference between the original images and their box filtered versions, and the color distance between those two images. It can be observed that the color distance image in the lower right corner is more robust in terms of overall brightness difference than its counterpart. It is still sensitive to high-frequency differences between the original source and target images.

vectors of two neighboring pixels are similar and the motion field is smooth. The smoothness constraint does not apply to edges of objects that move against a background and thus only piecewise smoothness can be postulated in the general case. Since our labeling prohibits using parts of photos that depict silhouettes or have steep viewing angles, the treatment of edges can be neglected here and it suffices to find a motion field that is globally smooth.

The algorithm outlined in this paper has been tested with three optical flow approaches, as proposed in [Dellepiane et al., 2012]. A naive brute force algorithm is discussed in section 3.1. An accelerated hierarchical version of the algorithm is explained in section 3.2. The implementation of the Floating Textures library is explained in section 3.3.

3.1 Brute Force Method

For this and the hierarchical method we assume that the displacement in image space is between -20 and 20 pixels both horizontally and vertically. This is the margin of error proposed in

[Dellepiane et al., 2012]. The input to the algorithm is one target image and one source image of equal sizes. The desired output is a warped version of the target image that is aligned with the features of the source image. For each pixel of the output image we calculate a vector that stores the relative position of the corresponding pixel in the target image.

It is important to keep in mind that the warping of the target image onto the source image is not a function, because pixels might be duplicated. Instead we calculate displacement vectors from the source to the target to ensure that every pixel in the output image has a target attached to it. Therefore we apply template matching as described in [Dellepiane et al., 2012], but leaving out the term f, that accounts for absolute brightness differences between the two images:

1. Around the currently evaluated pixel position $\mathbf{p}$, we construct the source template of 15×15 pixels in the source image and calculate its mean value $\bar{s}$.

2. We define a search space of 41×41 pixels around our current pixel in the target image. For each pixel position $\mathbf{x}$ in the search space, we do the following:

   a) We construct the target template of 15×15 pixels around $\mathbf{x}$ in the target image and calculate its mean value $\bar{t}$. This region can mapped directly onto the source template.

   b) We iterate over each pixel s_i in the source template and its corresponding pixel t_i in the target template and calculate the sum $S(\mathbf{x}) = \sum_i [(s_i - \bar{s}) - (t_i - \bar{t})]^2$, where each color channel is processed separately and the values added together. The result is a measure for the color difference between the two template areas. Overall brightness differences are thereby accounted for, as discussed in the previous section.

3. The displacement vector is the difference between $\mathbf{p}$ and the position $\mathbf{x}$ that minimizes the value of $S(\mathbf{x})$.

Formally, for each pixel at position $\mathbf{p}$, we are looking for a displacement vector $\mathbf{x}$ in the search space that minimizes the following term:

$$q(\mathbf{x}) = \sum_{c=0,1,2} \sum_{\mathbf{t}} g(\mathbf{p}+\mathbf{t}, \mathbf{x})_c^2 \tag{3.1}$$

$$\begin{aligned} g(\mathbf{z}, \mathbf{x})_c = &\left(I_S(\mathbf{z})_c - \bar{I}_S(\mathbf{z})_c\right) \\ &-\left(I_T(\mathbf{z}+\mathbf{x})_c - \bar{I}_T(\mathbf{z}+\mathbf{x})_c\right) \end{aligned} \tag{3.2}$$

$I_S(\mathbf{p})_c$ denotes the intensity of the source image for color channel c at position $\mathbf{p}$. $I_T(\mathbf{p})_c$ denotes the intensity of the target image for color channel c at position $\mathbf{p}$. $\mathbf{t}$ is the displacement vector inside the template. It iterates through all vectors in $\{-t_{max}, \ldots, t_{max}\}^2$. $\bar{I}_i(\mathbf{p})$ is the average intensity of the template area centered at $\mathbf{p}$ of image i.
t_{max} was chosen as 7, resulting in a template area of 15×15. The size of the search space is chosen as 41×41. Therefore a displacement vector in $\{-20, \ldots, 20\}^2$ is calculated for each pixel. If the camera registration errors are are likely to exceed this limit, a brute force approach is not advised due to long computing time.

The algorithm is implemented in a GLSL fragment shader. For each pixel, the shader loops over all possible displacement vectors $\mathbf{x}$ and chooses the one with a minimum value of $q(\mathbf{x})$. Each component of the displacement vector is encoded in a color channel of the output texture by normalizing the range $\{-20,\dots,20\}$ to $\{0,\dots,255\}$. Regions that do not contain information in the input images are encoded as white.

Only minimizing $q(\mathbf{x})$ does not take any smoothness constraints into account. The algorithm performs well in high-frequency regions of an image, but flat regions produce a jagged displacement vector field. In order to smooth the field, a median filter with a large kernel is applied to the x- and y-components of the field separately. An example for a color-encoded vector field can be seen in Figure 3.2. Warped versions of the target image are shown in Figure 3.3.

To calculate the optical flow between two images of size 2128×1416 pixels, the brute force algorithm takes 3:40 to complete. The computing time increases linearly with the image area.

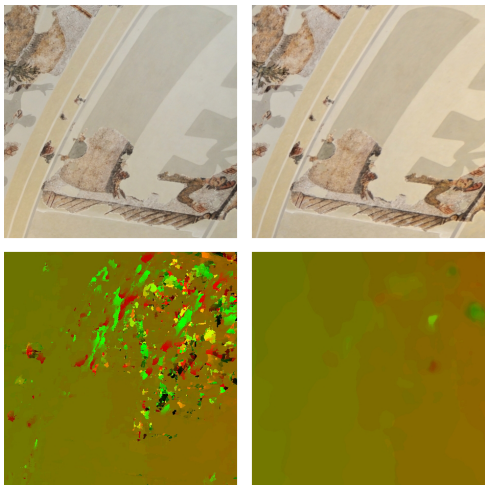

Figure 3.2: The top left and the top right images are target and source inputs. The bottom left image is the displacement vector field as produced by the brute force optical flow algorithm. The bottom right image is the displacement vector field after applying a median filter.

3.2 Hierarchical Method

The idea behind the hierarchical method is to calculate a displacement vector field between downsampled versions of the images first to restrict the size of the search space and thus accelerate the computation of the vector field between the original images. This method is proposed in [Namrata Vaswani, nd]. For this application, the following 2-level approach is used:

1. Create a downsampled version T' of the target T and a downsampled version S' of the source image S. The side length of the downsampled images is half the side length of the original images.

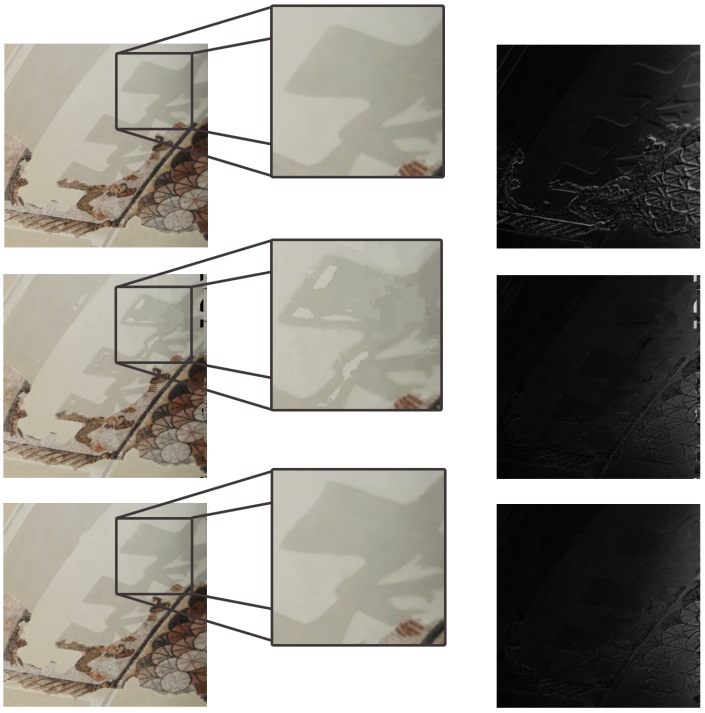

Figure 3.3: The first row shows the target image, a detailed view of a problematic region, and the color difference between the target and the source image to highlight the alignment errors. The second row shows the target image after it has been warped with the brute force optical flow algorithm. The color difference shows that the features have been aligned with the source image but monotonous areas exhibit strong artifacts. The third row shows the target image after it has been warped with the brute force optical flow + median algorithm. This avoids artifacts in flat areas.

2. Calculate a displacement vector field R' between T' and S' using the brute force approach with a 15×15 search space and a 5×5 template.

3. Upsample R', doubling its side length, to create a reference vector field R for the original target and source image.

4. Calculate the displacement vector field O between T and S. Use a 15×15 search space and a 5×5 template. The search space is centered around $\mathbf{p} + 2 * \mathbf{r}$. $\mathbf{p}$ is the position of the pixel that is currently evaluated. $\mathbf{r}$ is the displacement vector at position $\mathbf{p}$ in R.

The resulting displacement vectors are in the range of $\{-21, \ldots, 21\}^2$. This approach is significantly faster that the brute force algorithm because smaller search spaces are used. To calculate the optical flow between two images of size 2128×1416 pixels, the hierarchical algorithm takes approximately 5 seconds to complete.

A direct comparison between a warped image created by the brute force and hierarchical method can be seen in Figure 3.4. Both algorithms perform equally well in high-frequency regions. In flat regions, the hierarchical method produces significantly weaker distortion effects.

Figure 3.4: Left: target image. Center: warped image using brute force + median. Right: warped image using hierarchical + median. The center image has stronger distortions in low-frequency areas than the right image.

3.3 Floating Textures

The Floating Textures library provides an implementation of the methods proposed in [Eisemann et al., 2008]. The main focus is to create a dynamic mapping of multiple photographs onto a mesh at an interactive frame rate while avoid ghosting artifacts. Part of the library is a GPU implementation of the optical flow approach described in [Brox et al., 2004]. The following assumptions are made and deviations minimized to compute the flow field:

- Intensity constancy assumption: The intensity of a surface point stays the same over time.
- Gradient constancy assumption: Since intensity constancy is easily violated under changing lighting conditions, it is assumed that the spatial gradient of the intensity stays the same between two frames.
- Piecewise smoothness assumption: To deal with areas with a vanishing gradient, it is useful to assume that the flow field is smooth. Since silhouettes of moving objects against a background will result in discontinuities, only piecewise smoothness is postulated.

An advantage of this implementation is its high speed and accuracy under certain conditions. But the algorithm produces undesirable results if one photograph is slightly darker than the other. An illustration for this can be seen in Figure 3.5. If the source image is color-matched to the target image, the algorithm performs well in regions with high intensity gradients. The lower part of the image in Figure 3.5 is warped correctly. In flat regions with few discontinuities however, like the yellow stripes in Figure 3.5, the displacement vectors vanish and translation errors are not corrected.

For two input images with a resolution of 2128×1416 pixels, the algorithm takes less than 1 second to complete.

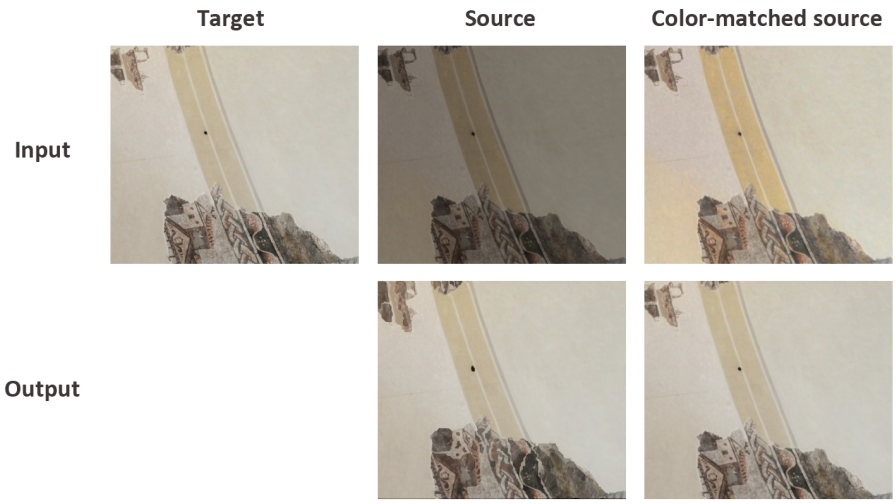

Figure 3.5: Left: target image. Center: original source image at the top, warped target at the bottom. Right: color-matched source image at the top, warped target at the bottom. The brightness difference between target and original source causes strong artifacts in the warped image. Color-matching between the target and the source avoids these artifacts.

CHAPTER 4

Stitching Algorithm

The stitching algorithm that is described in this section aims to correct discontinuity artifacts that appear along the edges between labels on a mesh. The findings of section 3 are used to create optical flow fields between neighboring labels. All examples in this section have been produced using the hierarchical approach.

The basis for our algorithm is the technique found in [Dellepiane et al., 2012]. The goal of their work is a stitching algorithm that blends suitable images while avoiding ghosting artifacts, which has been summarized in section 2.2. The key differences of our algorithm are the following:

- Our algorithm uses a partitioning where each label is assigned one photograph, rather than two. This leads to a strongly reduced number of labels and thus to fewer optical flow fields and shorter computing times.
- As input data, we use a partition with a small number of connected components and a short color distance between neighboring fragments along the seams. The implementation of the partitioning system is described in [Birsak, 2012] and has been provided by the courtesy of the author.
- We are not interested in blending different images while reducing ghosting artifacts, but warping them to minimize discontinuity artifacts.
- The warping function $\theta_k^{i,j}$ from the paper calculates the contribution of a fragment from image k to the color information in the label defined by images i and j. Since we do not require the computation of color contribution across label borders, we simplify the warping function by using a weighted sum of displacement vectors, as detailed below.

The following data are needed as inputs to the algorithm:

1. A mesh consisting of triangular faces

2. A set of photographs along with view and projection matrices

3. A labeling that maps each face onto one photograph

Preferably each label should contain large connected regions and few edges. If this is not the case, discontinuity artifacts are still eliminated but the photographs are distorted more than necessary in the process. The output of the algorithm is a new set of photographs that can be used with the same labeling and produces fewer artifacts. The algorithm could easily be adjusted to work on different geometry data like point clouds, but in this implementation triangular meshes are being used.

The first step is to analyze the labeling and find all ordered pairs $\{(L,N) \mid \text{L and N are labels and share a common edge}\}$. This relation is symmetric and all neighboring labels A and B will occur as (A,B) and (B,A). For every pair (L,N) the common edge $E_{L,N}$, a set of 3D line segments, is found.

The list of label pairs serves as the input to the next step. For each pair (L,N) we create an additional image N_L that shows the label N as seen from the perspective of L. This is done by rendering the mesh and transforming the vertices with the view and projection matrix of L. The photo of N is used as a projective texture, using the view and projection matrix of N to find the texture coordinates. Fragments whose texture coordinates are outside of $[0,1]^2$ are colored black. An illustration can be seen in Figure 4.1. To avoid coloring faces that are occluded by other faces, we use shadow volumes. The meshes that result from triangulating point clouds are usually not two-manifold, thus the traditional shadow volume generation approach is unusable. Instead, we employ the robust shadow volumes technique described in [Nguyen, 2007]. The image features of N_L overlap those of L, except for camera registration errors. Typically, N_L will look like a slightly translated version of L.

The next stage aims to correct these registration errors by calculating the optical flow field between every image pair (L,N_L). L serves as the target and N_L as the source image. The resulting displacement vector field O can be used to warp image L. The result matches the features of N_L perfectly. An example can be seen in Figure 4.2.

In order to apply the warp field only to the regions around the edges in a smooth way, these regions have to be defined. The goal is to adjust the length of the displacement vectors as seen in Figure 4.3. At the pixels that form the inside edge between two labels L and N, the displacement vectors are multiplied by 0.5. At a certain distance from the edge the vectors are multiplied by 0. In the space between, the multiplier of the vectors is linearly interpolated between 0 and 0.5.

Let us make this step more concrete. For each pair (L,N) we want to produce an image that contains the displacement vector multiplier for each pixel. This value should be 0.5 at the pixels on the edge and 0 at the pixels that are farther away than a certain distance d_{max}. The region in between, where the multiplier is not 0, is henceforth called the border of the seam. Therefore the set of line segments $E_{L,N}$ is rendered in black onto a white background in the image space of L. To the resulting binary image, a distance transform is applied. This process is illustrated in 4.4.

It is important to note that the photographs for both labels L and N must fully contain the border, otherwise the optical flow field does not contain sensible displacement vectors in that region. This constraint has to be taken into account during the labeling process and can be met

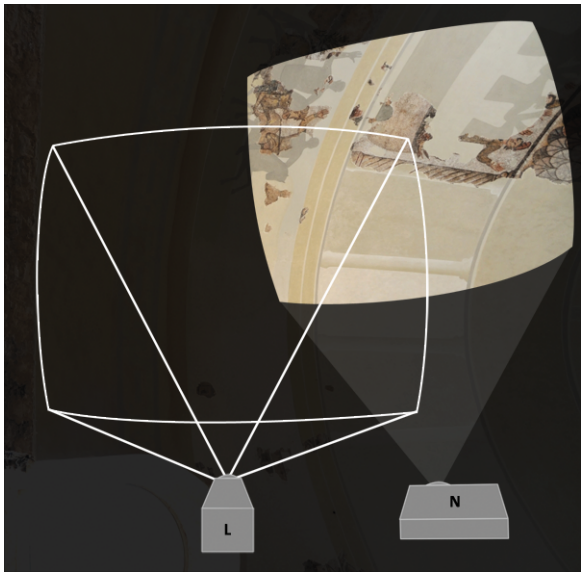

Figure 4.1: The label N is projected onto the mesh and transformed into the image space of L. The white frame contains the image N_L.

Figure 4.2: Left: Label L in its own image space. Center: Neighboring label N projected onto the mesh and inversely projected into L's image space, yielding N_L. Right: Optical flow field from L to N_L.

with a simple modification. Depending on the size of the border, the width and height of the view frustums are reduced before passing them to the labeling algorithm. For our test data, the frustum size was reduced by 10 percent in every direction. The effect is that the outer 10 percent of every photograph are not used for labeling. Therefore there is always an overlap between photographs of two neighboring labels.

The distances for each pixel are only valid in the image space of L. This means that the width of the border in model coordinates would depend on the pixel density of the photograph. To avoid this, the distance transform is corrected with respect to the depth of the pixels in the view space of L. We define d_{max} as the desired width of the border in pixels, if the camera's distance to the photographed surface is 1. This way the multipliers can be corrected according to the depth of the pixels.

To warp the texture for a label L, the corrected distance transform and the optical flow field

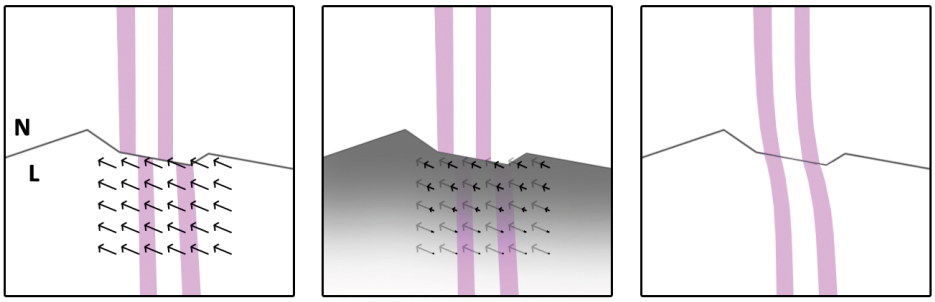

Figure 4.3: Left: A seam between two labels. The warp field maps the image features of L onto N. Center: The length of the displacement vectors is reduced the further a pixel is away from the seam. Right: The warp field is applied to L and the same procedure is used for N.

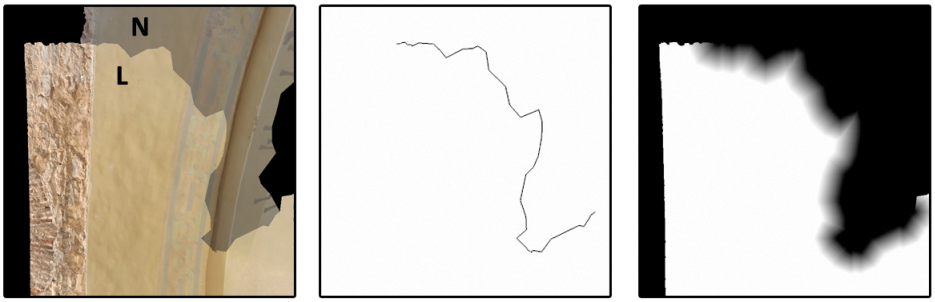

Figure 4.4: Left: Labels L and N in the image space of L. Center: The edge $E_{L,N}$ in black on white. Right: Distance transform of the (L,N).

must be produced for every neighboring label N_i, $i = 1,\ldots,k$. Then the resulting displacement vector for every pixel can be calculated as the weighted sum of each neighbor's displacement vector. The following algorithm is used:

```
// INPUT
p : position of the current pixel
DT[k] : corrected distance transforms
O[k] : optical flow fields
T : texture of L

// PROCEDURE
sumWarps := vec2(0,0)
sumWeights := 0

FOR(i from 1 to k)
    distance := 1.0 - sample(DT[i] at p)
```

```
    ofvector := sample(O[i] at p)

    weight := distance
    warp := ofvector * distance * 0.5

    sumWarps += warp * weight;
    sumWeights += weight;
END

IF (sumWeights > 0)
    p := p + sumWarps / sumWeights;
END

outColor := sample(T at p)
```

A GLSL implementation of the stitching algorithm is printed in Section 7.2.

CHAPTER 5

Results

The algorithm was tested on four data sets, obtained with two different laser scanner systems and mounted cameras.

- **Centcelles.** This data set was recorded with a RIEGL Laser Measurement System with a mounted digital single-lens reflex camera. The geometry consists of a triangulated model of the dome in the Mausoleum of Centcelles, that has been reduced to 300000 faces. The color information is provided by a set of photographs with 2128×1416 pixels each. The camera registration errors are clearly visible but do not exceed 20 pixels in any direction and therefore no problems are encountered when running the algorithm. Results in Figure 5.1.

- **Good Shepherd.** Acquired with a scanner from Adam Technology, the geometry consists of a triangulated model with 300000 faces of the Good Shepherd Dome. The photographs have a size of 4064×2704 pixels each. The misalignment errors in this data set are very small and often barely visible. Thus the main concern is that filtering errors introduced by the optical flow process must not be larger than the registration errors that are corrected. By using median filtering of the flow fields, this criterion is met. Results for this data set can be seen in Figure 5.2.

- **Terrace House 2.** The Terrace House 2 in Ephesus was scanned using a RIEGL scanner and produced a mesh of more than 15 million faces after triangulation. The photographs have a resolution of 4256×2832 pixels and contain only small registration errors. To test whether the algorithm works well with imprecise meshes, the model was reduced to 500000 faces, yielding a very coarse approximation of the original. Similar to the Good Shepherd results, the registration errors do not exceed a few pixels but these errors are successfully corrected. Results in Figure 5.3.

- **Seven Sleepers cave.** The Seven Sleepers cave in Ephesus was scanned using a RIEGL scanner and reduced to 500000 faces. The photographs are 4256×2832 pixels large,

	Centcelles	Good Shepherd	Terrace House 2	Seven Sleepers cave
Number of labels	6	6	6	12
Number of pairs	20	16	12	56
Number of faces	300k	300k	500k	500k
Image dimensions	2128×1416	4064×2704	4256×2832	4256×2832
Label generation	1m 10s	1m 15s	2m 10s	4m 40s
Model processing	10s	15s	45s	30s
Flow field generation (H)	1m 15s	3m 0s	2m 25s	10m 25s

Table 5.1: Results for the labeling, optical flow field generation and stitching process.

exhibit high brightness differences and many are underexposed or blurry. Results are shown in Figure 5.4.

A labeling was produced for each data set using the MRF implementation without shift vectors from [Birsak, 2012]. Computing times for the steps of the algorithms are provided in Table 5.1. For the optical flow field calculation, we used NVIDIA GeForce 570 GTX hardware. The generation of N_L for a pair (L,N) and the stitching of a label by using the flow fields can be done with one shader call each. The stitched textures may be generated once and then reused, but the stitching algorithm is also quick enough to operate on the original textures and the flow fields in real-time.

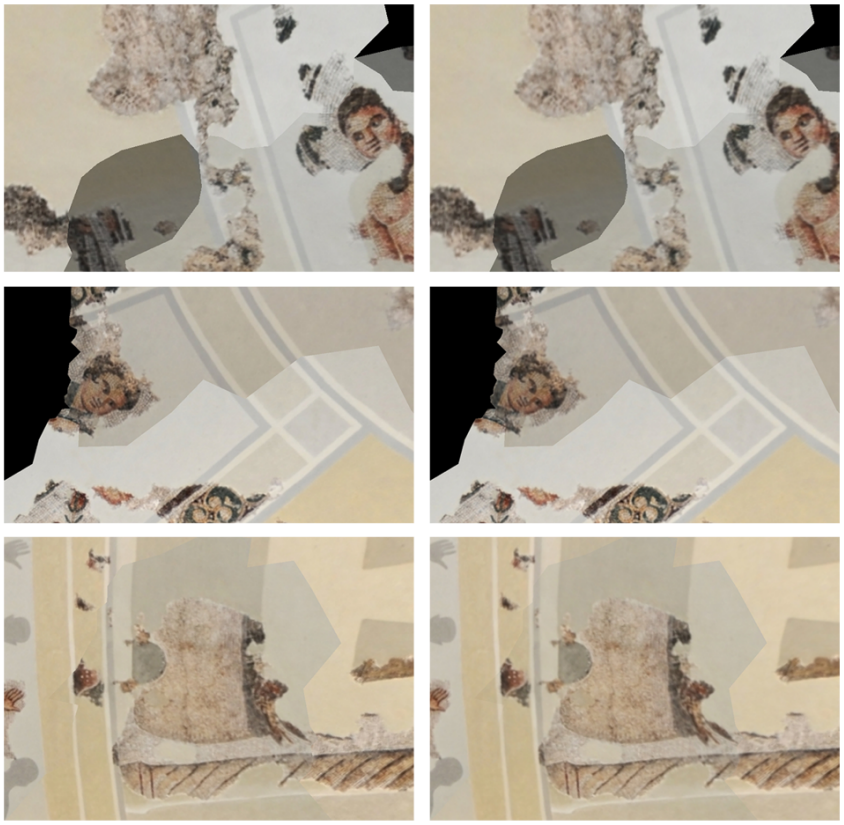

Figure 5.1: Comparison between labeled data sets with and without seam stitching. Images from the Centcelles data set.

Figure 5.2: Comparison between labeled data sets with and without seam stitching. Detail view of seams in the Good Shepherd data set.

Figure 5.3: Comparison between labeled data sets with and without seam stitching. Images from the Terrace House 2 data set. The contrast of the photos has been adjusted to make the improvement visible.

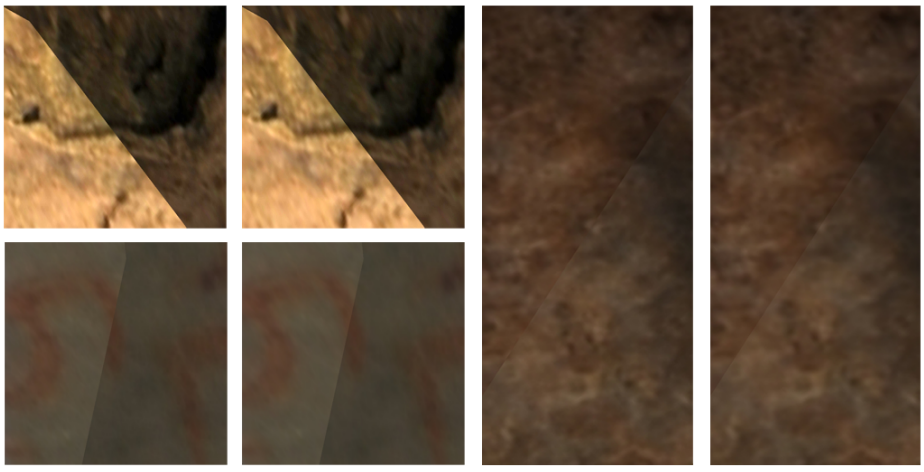

Figure 5.4: Comparison between labeled data sets with and without seam stitching. Images from the Seven Sleepers data set. The contrast of the upper left photos has been adjusted to make the improvement visible.

CHAPTER 6

Conclusion

This thesis presents a fully automatic approach to effectively reduce discontinuity artifacts in an image-to-geometry setting. We evaluate existing labeling and optical flow strategies and combine suitable methods to form a stitching algorithm that is based on local optimization. Our algorithm takes a labeled triangle mesh as input data to find neighboring labels. Each label is projected into the image space of its neighbor and an optical flow field between them is estimated. These fields are applied at the border between labels to remove discontinuity artifacts.

The algorithm has been shown to be computationally feasible and to generate high-quality results with no manual parameterization required. It scales linearly with the number of neighboring photographs and is robust in terms of lighting conditions and image quality. The labeling and optical flow stages can easily be substituted with with other procedures that are suitable to the particular application. Future works could explore the compatibility of the results with leveling approaches in order to reduce remaining brightness discontinuities. Also the optical flow estimation could be combined with the stitching stage such that displacement vectors are only calculated in the border region between photographs, thus speeding up the process.

CHAPTER 7

Listings

7.1 Brute force Optical flow shader

```
#version 330 core

uniform sampler2D target; // search space
uniform sampler2D source; // template
uniform vec2 imageSize; // in pixels (width, height)
uniform int lod; // mipmap of target and source image, 0 for original

in vec2 vf_texCoords;

out vec3 colorCode;

void main(void)
{
  // early out if no image information
  if (textureLod(target, vf_texCoords, lod).xyz == vec3(0.0,0.0,0.0))
  {
    colorCode = vec3(1.f,1.f,1.f);
    return;
  }
  // size of one pixel
  vec2 invImageSize = vec2(1.0/imageSize.x, 1.0/imageSize.y);
  vec2 currentTexCoords;

  // calculate mean color intensities of template in source
  vec3 meanSource = vec3(0.0,0.0,0.0);

  int runningIndex = 0;
  for (int y = -7; y <= 7; y++)
  {
    for (int x = -7; x <= 7; x++)
    {
      currentTexCoords = vf_texCoords + vec2(invImageSize.x * x, invImageSize.y * y);
      meanSource += textureLod(source, currentTexCoords, lod).xyz;
    }
  }
```

```
  meanSource /= 225.0;

  vec2 bestPos;
  float minDif = 9999999999.0;

  // Traverse through search space
  for (int y = -20; y <= 20; y++)
  {
    for (int x = -20; x <= 20; x++)
    {
      // calculate fit value between template in source
      // and template in search space
      float curDif = 0;

      // calculate mean color intensities of current
      // area in search space
      vec3 meanTarget = vec3(0.0,0.0,0.0);
      for (int pY = -7; pY <= 7; pY++)
      {
        for (int pX = -7; pX <= 7; pX++)
        {
          currentTexCoords = vf_texCoords + vec2(invImageSize.x * (x+pX), ↩
              invImageSize.y * (y+pY));
          meanTarget += textureLod(target, currentTexCoords, lod).xyz;
        }
      }

      meanTarget /= 225.0;

      // calculate value of q(x)
      for (int pY = -7; pY <= 7; pY++)
      {
        for (int pX = -7; pX <= 7; pX++)
        {
          currentTexCoords = vf_texCoords + vec2(invImageSize.x * pX, invImageSize.y ↩
              * pY);
          vec3 sourcePick = textureLod(source, currentTexCoords, lod).xyz;
          currentTexCoords = vf_texCoords + vec2(invImageSize.x * (x+pX), ↩
              invImageSize.y * (y+pY));
          vec3 targetPick = textureLod(target, currentTexCoords, lod).xyz;
          vec3 dif2 = (sourcePick - meanSource) - (targetPick - meanTarget);
          curDif += dot(dif2,dif2);
        }
      }

      // choose displacement vector with minimal q(x)
      if (curDif <= minDif)
      {
        minDif = curDif;
        bestPos = vec2(x, y);
      }
    }
  }

  // encode best displacement vector
  colorCode = vec3((bestPos.x+20)/40.0, (bestPos.y+20.0)/40.0, 0.0);
}
```

7.2 Stitching shader

```
uniform sampler2D diffuseTexture; // photo for the label
uniform sampler2D distanceTransform[6]; // distance transforms for all edges
uniform sampler2D warpField[6]; // optical flow fields for all neighbors
uniform int numNeighbors; // number of neighbors
uniform vec2 imageSize; // image size in pixels
uniform int maxTranslation; // maximum value of displacement vector components

in vec2 vf_texCoords; // texture coordinate of current fragment

out vec3 outColor;

void main(void)
{
  vec2 invImageSize = vec2(1.0 / imageSize.x, 1.0 / imageSize.y);

  // will contain weighted blending of displacement vectors
  vec2 sumWarps = vec2(0.0,0.0);
  float sumWeights = 0.0;

  for (int i=0; i<numNeighbors; i++)
  {
    // color-coded displacement vector
    vec3 of3 = texture(warpField[i], vf_texCoords).rgb;

    // white means that the photo is not overlapping with its
    // neighbor on this pixel
    if (of3 == vec3(1.0,1.0,1.0))
    {
      continue;
    }

    // calculate displacement vector
    vec2 of = of3.rg;
    of = of * 2.0 * float(maxTranslation) - vec2(maxTranslation,maxTranslation);

    // adjust weight based on the distance between the pixel and the edge
    float correctDistance = 1.0 - texture(distanceTransform[i], vf_texCoords).r;

    // add weight adjusted displacement vector to total
    float weight = correctDistance;

    sumWarps += weight*of*invImageSize*correctDistance*0.5;
    sumWeights += weight;
  }

  if (sumWeights > 0)
  {
    // new texture coordinate after adding total displacement vector
    vec2 newTexCoords = vf_texCoords+(sumWarps/sumWeights);
    outColor = texture(diffuseTexture, newTexCoords).xyz;
  }
  else
  {
    outColor = texture(diffuseTexture, vf_texCoords).xyz;
  }
}
```

Bibliography

[Birsak, 2012] Birsak, M. (2012). Coloring meshes of archaeological datasets. Master thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

[Brox et al., 2004] Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. (2004). High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision (ECCV)*, volume 3024 of *Lecture Notes in Computer Science*, pages 25–36. Springer.

[Callieri et al., 2008] Callieri, M., Cignoni, P., Corsini, M., and Scopigno, R. (2008). Masked photo blending: mapping dense photographic dataset on high-resolution 3d models. *Computer & Graphics*, 32(4):464–473. for the online version: http://dx.doi.org/10.1016/j.cag.2008.05.004.

[Dellepiane et al., 2012] Dellepiane, M., Marroquim, R., Callieri, M., Cignoni, P., and Scopigno, R. (2012). Flow-based local optimization for image-to-geometry projection. *Visualization and Computer Graphics, IEEE Transactions on*, 18(3):463–474.

[Eisemann et al., 2008] Eisemann, M., De Decker, B., Magnor, M., Bekaert, P., de Aguiar, E., Ahmed, N., Theobalt, C., and Sellent, A. (2008). Floating textures. *Computer Graphics Forum (Proc. of Eurographics)*, 27(2):409–418. Received the Best Student Paper Award at Eurographics 2008.

[Gal et al., 2010] Gal, R., Wexler, Y., Ofek, E., Hoppe, H., and Cohen-Or, D. (2010). Seamless montage for texturing models. *Comput. Graph. Forum*, pages 479–486.

[Lempitsky and Ivanov, 2007] Lempitsky, V. and Ivanov, D. (2007). Seamless mosaicing of image-based texture maps. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1 –6.

[Lensch et al., 2000] Lensch, H. P. A., Heidrich, W., and peter Seidel, H. (2000). Automated texture registration and stitching for real world models. In *in Pacific Graphics*, pages 317–326.

[Namrata Vaswani, nd] Namrata Vaswani (n.d.). http://home.engineering.iastate.edu/ namrata/ee520/opticalflow.pdf. Accessed: 2013-05-01.

[Nguyen, 2007] Nguyen, H. (2007). *Gpu gems 3*. Addison-Wesley Professional, first edition.

[T. Takai and Matsuyama, 2010] T. Takai, A. H. and Matsuyama, T. (2010). Harmonised texture mapping. In *Proc. 3D Data Processing Visualization and Transmission (3DPVT '10)*.