Automated Texture Registration and Stitching for Real World Models

Hendrik P. A. Lensch Wolfgang Heidrich Hans-Peter Seidel
Max-Planck-Institut für Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany.
{lensch,heidrich,hpseidel}@mpi-sb.mpg.de

Abstract

In this paper a system is presented which automatically registers and stitches textures acquired from multiple photographic images onto the surface of a given corresponding 3D model. Within this process the camera position, direction and field of view must be determined for each of the images. For this registration, which aligns a 2D image to a 3D model we present an efficient hardware-accelerated silhouette-based algorithm working on different image resolutions that accurately registers each image without any user interaction. Besides the silhouettes, also the given texture information can be used to improve accuracy by comparing one stitched texture to already registered images resulting in a global multi-view optimization. After the 3D-2D registration for each part of the 3D model's surface the view is determined which provides the best available texture. Textures are blended at the borders of regions assigned to different views.

1. Introduction

Throughout the past years 3D rendering solutions have advanced in rendering speed and realism. Because of this, there is also an increased demand for models of real world objects, including both the object's geometry and its surface texture. Precise geometry is typically acquired by specialized 3D scanners while detailed texture information can even be captured by consumer quality digital cameras. Only a few 3D scanning devices are built to capture 3D geometry and 2D textures at the same time. And even if texture acquisition is supported it may be required to take the images under controlled lighting conditions with a special sensor implying that the object of interest has to be placed in a fully controllable environment while taking the pictures. In cases where photos and geometry are not acquired by the same sensor, the images must be registered with the 3D model afterwards in order to connect geometry and texture information.

For this registration task we present a hardware-accelerated algorithm that aligns an image to the 3D model as well as to other already registered images. All stages of the algorithm can run completely automatically. Alternatively, the user can skip some steps in the algorithm providing a rough alignment.

2. Related Work

In the field of capturing surface appearance (color and texture) of real world objects there have been a number of recent publications ranging from architectural scenes [3, 21] to smaller artifacts [9, 10, 12, 16, 17] and even deformable objects like faces [5, 6, 13]. To acquire a complete texture for an object the following tasks must be performed.

2.1 Imaging All Visible Surfaces

If an object's surface should be entirely digitized the first step is to collect data for all visible surfaces. A set of camera positions must be determined from which every part of the surface is captured by at least one image. For a given geometric model and a set of possible positions Matsushita et al. [10] determine the optimal set of required views respecting the viewing angle. Further, Stuerzlinger [19] finds a minimal set of view points within the volume of all possible camera positions. He uses hierarchical visibility links to first determine optimal subregions using a simulated annealing approach, and then selects optimal points within these regions.

2.2 3D–2D Registration

After taking the images the camera position and rotation relative to the 3D model must be determined for each view. Only if geometry and texture are acquired at the same time with the same sensor like in [17] or [15], the images are already aligned to the model and no further 3D–2D registration is needed. In all other cases, one can basically follow two different approaches.

The first approach selects a set of points in each image which correspond to known points on the model's surface. From these correspondences the camera transformation for the current view can be directly derived using standard camera calibration techniques, e.g. [20]. However, the problem is to find these pairs of points. Depending on the object there may be geometric feature points which can be easily located in the images, and thus can be detected and assigned automatically. Kriegman et al. [7] use T-junctions and other image features to constrain the model's position and orientation. Others attach artificial landmarks to the object's surface which are detected automatically in the images [5]. But these marks destroy the texture and have to be removed afterwards. If no extraordinary points can be detected automatically one may of course select corresponding pixels manually, which actually is a commonly used but tedious method [13, 16, 3].

Instead of directly searching for 3D–2D point pairs, one may inspect larger image features like the contours of the object within each image. The correct camera transformation will project the 3D model in such a way that the outline of the projected model and the outline in the image match perfectly except for small errors due to imprecise geometry acquisition.

A lot of previous algorithms try to find the camera transformation by minimizing the error between the contour found in the image and the contour of the projected 3D model [2, 8, 12, 10, 6]. The error is typically computed as the sum of distances between a number of sample points on one contour to the nearest points on the other [12, 10]. Another approach computes the sum of minimal distances of rays from the eye point through the image contour to the model's surface which are computed using 3D distance maps [2].

To recover the different camera parameters, any kind of non-linear optimization algorithm like Levenberg-Marquardt, simulated annealing, or the downhill simplex method can be used (see [14] for an overview). During the optimization a lot of different settings for the camera parameters are tested in order to guide the algorithm towards a minimum. For each test the error function has to be evaluated which is quite costly for contour-based distance measurement since the model must be projected and the point distances to the projected contour must be calculated for a sufficient number of points. In Section 5 we present a different, more efficient algorithm to calculate the distance between silhouettes instead of contours.

Beside geometry-based 3D–2D registration, the texture/image information itself may be used to register the different views relative to each other. For 2D–2D image registration a number of techniques have been developed [1]. Based on this pairwise registration a global optimization for all incorporated views can be performed as demonstrated by Neugebauer et al. [12], whereas Rocchini et al. [16] use the image information only to align the textures in those regions where different textures have to be blended during rendering.

2.3 Texture Preparation and Rendering

After registration the mapping from surface parameters to texture coordinates is known for each view. A single image can be mapped onto the object by common graphics hardware supplying projective texture mapping [18]. If multiple views are incorporated one must determine which image is best to be mapped onto which part of the surface. Here, the angle between the viewing direction during acquisition and the surface normal may be considered [16, 10], or the textures are selected depending on the rendering view point [3, 4, 15]. Special care must be taken at boundaries of surface regions which are textured with data from different images. To create a smooth transition between the regions the textures must be blended appropriately. Rocchini *et al.* [16] even precomputed this blending into a new texture to speed up the entire rendering process. Additionally, all relevant parts of the original images are packed into one single large texture to provide easier handling.

3. Overview / Contributions

Out of the set of the different tasks necessary to acquire a complete texture mentioned in the previous section, we present new solutions for the following ones:

- single view registration based on silhouettes (Section 5 and Section 6)
- global registration of multiple views with respect to image features (Section 8)
- view-independent assignment of surface parts to the images providing the best texture for the single part (Section 7)

- blending between textures at assignment boundaries (Section 7)

Although we briefly explain all necessary steps from image acquisition to rendering of the textured model, the main focus within this paper is on novel techniques for image registration.

4 Camera Transformation

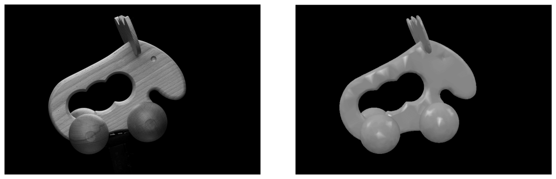

Figure 1. Recovering the camera parameters for one image allows to map the image correctly onto the model.

During registration the camera settings must be determined for each image mapping it correctly onto the 3D model (Figure 1). In our system a pinhole camera model is assumed. Up to seven camera parameters are recovered: the field of view which is the only intrinsic parameter and is related to the focal length, and six extrinsic parameters describing the camera pose and orientation. All other intrinsic parameters like aspect ratio, principal point, or radial lens distortion are assumed to be constant and known since they can be obtained easily using common camera calibration tool kits, or they are simply ignored and set to reasonable approximative values.

The camera position is expressed by the translation vector $t_c \in \mathbb{R}^3$, while the orientation of the camera is described by (ϕ_x, ϕ_y, ϕ_z), the rotation angles about the coordinate axes, which form a 3×3 rotation matrix R. These extrinsic parameters determine a rigid body transformation that maps a point in world coordinates $x_w \in \mathbb{R}^3$ into camera coordinates $(x_c, y_c, z_c)^T$:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = Rx_w + t_c \tag{1}$$

For a camera far away from the object this representation has the disadvantage that a small rotation around the camera results in a large displacement of the object in camera coordinates. If the point x_w is rotated around the center of gravity g of the object instead the effects of rotation and translation are much easier to distinguish, thus simplifying the optimization [12]. The translation is now given by $t = Rg + t_c$ which actually is the position of the center of gravity in camera coordinates.

$$\begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = R(x_w - g) + Rg + t_c = R(x_w - g) + t \tag{2}$$

To fully describe the camera transformation the points $(x_c, y_c, z_c)^T$ are further mapped to 2D image space (u, v):

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \frac{1}{z_c} \begin{pmatrix} \frac{f}{\alpha} x_c \\ f y_c \end{pmatrix}, \tag{3}$$

where (u_0, v_0) is the principle point (in our case the center of the image), α the aspect ratio of width to height which must be provided by the user, and f the field of view. Thus, the camera transformation is determined by f and the vector $\pi = (\phi_x, \phi_y, \phi_z, t_x, t_y, t_z)^T$. For each image these seven parameters have to be recovered by a non-linear optimization of a similarity function comparing the projected model to the object found in the image.

5. Similarity Measure

Since we want to optimize seven parameters $(\phi_x, \phi_y, \phi_z, t_x, t_y, t_z, f)$ we define a function $s : \mathbb{R}^7 \rightarrow \mathbb{R}$ which returns a scalar value for the specified camera transformation expressing the similarity of the projected model and an image, i.e. with a small value indicating high similarity while the value increases when the projected model and the image are misaligned. As this function s will be evaluated quite often during the optimization process it is necessary that it can be computed very quickly.

At first we have to define in which way we want to measure the similarity, which feature space to be used. Since the 3D geometric model does not yet carry any color information we are restricted to geometric properties. In contrast to Neugebauer et al. [12] and Matsushita et al. [10] who compared the contour of the projected model to the contour in the image, we decided to directly compare the silhouettes, which requires less computation. A silhouette is the object projected onto a plane filled with uniform color while a contour is the outline of the silhouette.

5.1 Segmentation

When rendering the model for a given view the silhouette can be generated simply by choosing a uniform white color in front of a black background which is very simple. If instead of the silhouette the contour had to be extracted further processing would be necessary which we can avoid.

To compare silhouettes the second silhouette must be extracted from the image data. If the object is captured in front of a black background the image can be segmented automatically by histogram-based thresholding. The threshold is chosen right after the first peak in the histogram which corresponds to the number of very dark pixels. If the contrast between the object and the background is too low (like in less controllable environments) other image processing techniques must be applied. For example the semi-automatic algorithm presented by Mortensen and Barret [11] may be used to trace the contour in the image which afterwards can be filled automatically. This segmentation has to be done only once for each image before starting the actual optimization and thus user interaction seems acceptable.

5.2 Silhouette Comparison

After extracting the silhouettes some kind of distance measurement between the silhouettes has to be defined. The technique presented here can be carried out completely by use of commonly available graphics hardware supporting histogram evaluation.

The first step renders the silhouette of the projected 3D model into the framebuffer. The result is then combined with the segmented image using a per-pixel XOR-operation. This process is visualized in Figure 5 where the silhouettes are computed for the photo and for one view of the 3D model and combined afterwards. After the XOR-operation exactly the pixels between the outlines remain white. Their number can be counted by simply evaluating the histogram of the combined image which is computed very efficiently by the graphics hardware. For exact matches a value close to zero will be returned while the number of remaining pixels will be much larger if the rendered view of the model is different from that in the photo.

The computation time for the similarity function is dominated by two quantities. The more important one is the resolution selected for rendering since each pixel of the XORed image will be processed during the computation of the histogram. The other quantity is the complexity of the 3D model in terms of the number of geometric primitives that have to be rendered to produce the model's silhouette.

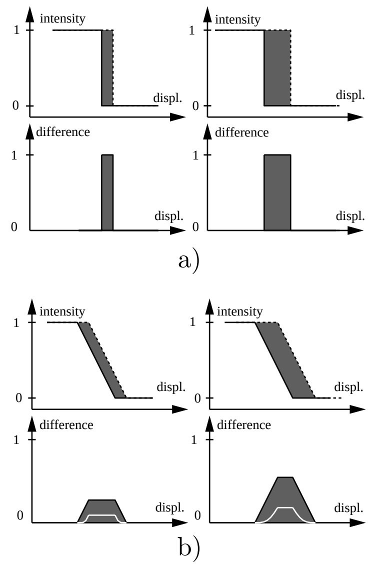

Figure 2. a) The integral of differences between a sharp intensity edge and the same edge slightly displaced (dashed) is proportional to the displacement. b) Blurred edges also produce a linear distance measure. But the differences between blurred edges can be squared before integration approximating a quadratic measurement (white line).

5.3 Blurred Silhouettes

Until now, we have assumed monochromatic silhouette images with a sharp transition between the intensity of pixels belonging to the object and those belonging to the background. Suppose two sharp intensity transitions which are slightly displaced like depicted in Figure 2a. As the displacement is increased, the integral of the differences of the two curves grows linearly while the differences are either one or zero. This is exactly the result of the presented similarity measurement based on XORed monochromatic silhouettes.

More desirable is a measurement that is proportional to the squared distance between points on the outlines. This behavior can be approximated for small displacements using blurred edges. As can be seen in Figure 2b, even for blurred transitions the integral of the differences between the curves is proportional to the displacement. But in this case also the magnitude of the differences is linear to the displacement in regions where the transitions overlap. These differences can be squared prior to the integration. By this, a quadratic distance measurement is approximated for edges as long as the displacement of the edges is smaller than the size of the filter kernel applied to blur the edges. Larger displacements are emphasized compared to smaller ones. This behavior can guide the optimization algorithm faster to the minimum. But computing the differences between blurred images is slightly more expensive than just applying the XOR-operation and one can decide if it is worth the cost (see Section 9).

To blur the silhouettes a $n \times n$ low-pass filter is applied. While this is no problem with respect to the photo since it is done before the optimization, the silhouette of the projected 3D model must be filtered again for each view. Although convolution can be computed by the graphics hardware, it requires processing the entire framebuffer and thus slows down the evaluation of the similarity function. After blurring the silhouettes the absolute difference values between them must be computed on a per-pixel basis. A special OpenGL extension allows to compute the positive difference of the framebuffer contents and an image by specifying a particular blending equation. Since only positive values are computed while negative values are clamped against zero we first render the silhouette of the 3D model minus the photo into the red channel and then the photo minus the 3D model's silhouette into the green channel of the framebuffer as can be seen in Figure 6. The histogram of the red and of the green channel are then combined to obtain the sum of the absolute values, and the approximate quadratic distance is computed.

5.4 Erroneous Pixels

For real photos the defined similarity function will always return values much larger than zero no matter how close the determined view comes to the original view of the photo. There are always some pixels of the silhouette in the photo which are not covered by the projected 3D model or vice versa, originating from different sources of error. On one hand the 3D model may be somewhat imprecise due to the acquisition. There may be even parts of the object visible in the image which are not part of the 3D model. On the other hand some pixels in the image may be wrongly classified by the automatic segmentation due to unfavorable lighting conditions. Additionally, in some views parts of the object will be hidden by other objects.

There are several possible ways to deal with these erroneous pixels: If the regions of erroneous pixels do not penetrate the silhouette of the object like holes within the silhouette or bright regions in the image apart from the object (Figure 3a) the optimization is not affected since these pixels only add a constant bias to the histogram. If erroneous pixels disturb the outline they may lead to slight misregistration (Figure 3b). But the error may be corrected afterwards by comparing the registered texture to the texture of other views as explained in Section 8, or it may simply be ignored if it is only small. In cases where the error is too large to be acceptable the erroneous pixel can be masked out and the histogram is evaluated only over regions providing reliable information. But masking out the bad regions requires user interaction and thus should be avoided.

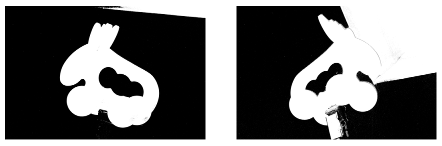

Figure 3. Large regions of wrongly segmented pixels apart from the silhouette (a) and penetrating the silhouette (b).

6. Non-linear Optimization

Let us assume a similarity function s as defined in the previous section. To recover the correct transformation for a given image we have to find the pair (π_{min}, f_{min}) that minimizes s. Since s typically is non-linear and possesses a bunch of local minima, an appropriate optimization method must be applied. We chose the *downhill simplex method* as it is presented in [14] and extended it by some aspects similar to simulated annealing since the pure simplex method tends to converge too fast into local minima. Of course, also other optimization techniques may be used instead but we found the simplex method easy to control and it does not require any partial derivatives, which makes it very efficient, even if the cost for evaluating the similarity function is high, like in our case.

6.1 Downhill Simplex Method

The method works for N-dimensional problems albeit we use it only for 6 dimensions to optimize the camera pose and orientation. The field of view is optimized afterwards using another optimization technique since a good approximation can be derived from the applied lens, and further, the effect of changing the field of view is too similar to changing the distance making the optimization less stable.

Starting with N points spanning a simplex in $\mathbb{R}^N$, the simplex method sorts the points depending on their similarity function values and selects the worst point p_{hi} with the greatest $s(p_{hi})$. The method then

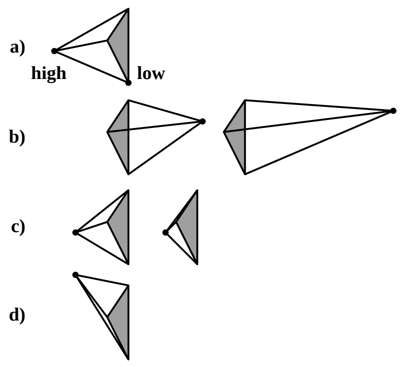

Figure 4. Possible results after one optimization step: a) The initial simplex. b) The high point is reflected and perhaps further expanded. c) Contraction along one dimension from the high point. d) The high point is perturbed randomly.

tries to find a new p' with $s(p')$ less than $s(p_{nhi})$ of the next better point by testing a set of positions in sequential order [14]. These are depicted for three dimensions in Figure 4: p_{hi} is reflected through the opposite face of the simplex and even further displaced in the same direction if a better result can be achieved (Figure 4b), p_{hi} is pulled towards the center of the simplex which may even be repeated (Figure 4c). At last, p' is set randomly within an adaptive radius (Figure 4d) if the position could not be improved by one of the preceding steps. Instead of a random displacement Press et al. [14] propose a contraction of all points towards the best point which unfortunately makes the method converge too fast to a local minimum. After improving p_{hi}, sorting, selection and improving are repeated until the radius of the simplex is less than some user-defined threshold.

6.2 Hierarchical Optimization

The algorithm still converges very quickly to a minimum that is not necessarily the global minimum. In order to find the global minimum we restart the optimization process several times. Hereby, the minimum found by the previous optimization is used as the next starting point. The radius of the initial simplex is of course reduced before each iteration to speed up the convergence.

Another method to speed up the optimization and even to increase robustness is to run the optimization at different image resolutions. As pointed out in Section 5 the evaluation time for our similarity measurement depends on the used image resolution since the histogram has to count all pixels. Starting with low resolution the view can be approximated roughly but very quickly. For accurate registration the resolution is increased. At the same time also the tessellation of the object can be varied to gain a speedup.

6.3 Generating a starting point

For the optimization it is important to have an appropriate starting point. A starting value for the field of view can be derived directly from the focal length of the applied lens which is reported by some digital cameras. This typically won't be the correct focal length since it is slightly changed by selecting the focal distance. Assuming that the entire object is visible in the image, an initial guess for the distance can be computed using the field of view and the size of the object. The x and y displacements are initially assumed to be neglectable.

What remains is to make a guess for the orientation. This is done by sparsely sampling the space of possible angular directions. We try three different angles for ϕ_x and four for both ϕ_y and ϕ_z yielding 48 samples. From each of these samples we start the simplex algorithm running at a rather low resolution and stop already after a few evaluations of the similarity function. The best five results are selected and further optimized, this time allowing more evaluations at the same resolution. It turned out that the best of the computed minima is already quite close to the one we are searching for. With this value the final optimization can be started.

Of course the generation of the starting point takes some time, but it does not require any user interaction. Especially, there is no need to select pairs of corresponding points. However, time can be saved by manually moving and rotating the 3D model very roughly into a position similar to the photo.

For a fixed field of view the following steps are performed to recover the translation vector t and the rotation R given by ϕ_x, ϕ_y, ϕ_z: generate a starting position automatically like described above or select it manually, run the simplex method two times at low image resolution and then two times at the final resolution.

6.4 Optimizing the Field of View

Given the optimized π, the field of view f_{start} obtained from the camera and the result of the similarity function $s(\pi, f_{start})$ we now try to find the best field of view f_{min} that further minimizes $s(\pi', f_{min})$ where π' is only slightly changed compared to the previous π. This problem is a search in only one dimension for which we implemented a simple algorithm.

Let us start with f set to f_{start}. At first f is increased by an amount d yielding a new f'. All other parameters are simply copied from π to π'. Then the distance t_z in π' is updated to compensate for the change in the field of view in such a way that the size of the projected object approximately remains the same for the new f'. To π' and f' the simplex method is applied allowing only a few evaluations of the similarity function. This yields an optimized parameter set π'_{opt}. This optimization is necessary to slightly correct π' since a wrong field of view will lead to a wrong registration in the other parameters too. If by increasing f a better field of view was found $(s(\pi'_{opt}, f') < s(\pi, f))$ the field of view is increased and the algorithm is repeated, starting with (π'_{opt}, f'). Otherwise we divide the increment d by two, step back to the predecessor of the last field of view and proceed with the search until d is sufficiently small. If no better field of view can be found by increasing f_{start} the algorithm is just applied into the other direction, decreasing f_{start}.

Using this algorithm it is possible to determine the best field of view for each photograph independently. This allows to select a different focal distance or even different lenses for each view in contrast to previous approaches in which the field of view had to be fixed [12, 10].

7. Texture Stitching

After determining the correct viewing parameters for an image, it can be stitched as a texture onto the surface of the 3D model. In this section a triangular mesh is assumed although the presented ideas can easily be adapted to other surface representations as well.

7.1 Single View Processing

Given the viewing transformation the set of visible vertices of the 3D model can easily be determined either by casting a ray from the view point to the vertex and testing for occlusion or by a simple z-buffer depth test. For all visible vertices a texture coordinate into the image is computed by projecting the vertex into the image plane using the recovered camera transformation. Additionally, the viewing angle is determined for each vertex. From this data the set of usable vertices is derived. A vertex is declared valid only if the viewing angle at that point is large enough, the depth variation around that point is not too steep and the point does not lie exactly on the outline of the projected object. Using this criterion texture mapping artifacts can be avoided when viewing the textured object from views than the determined one.

Based on the set of valid vertices those triangles can be selected for which reliable texture information is available. A triangle is used only if all its vertices are valid.

7.2 Combining Multiple Textures

If multiple images are involved, the sets of valid triangles will overlap and the best assignment of triangles to images must be determined. A static decision can be made by inspecting again the angle under which the triangle is seen in each image. Each triangle is assigned the texture from that image in which it possesses the largest viewing angle.

There will be triangles that are assigned to one image while an adjacent triangle is assigned to another image (Figure 7a). This often results in a visible discontinuity in the texture even if the images are taken without changing the lighting conditions. A smooth transition is achieved by blending between the textures across the border triangles. This requires all boundary triangles to be valid also for adjacent textures. To ensure this the set of valid triangles for each image is reduced prior to the assignment to the images. All those triangles are invalidated which have at least one invalid triangle as their neighbor.

Next, the triangles must be determined across which to blend. All triangles containing a boundary vertex are possible candidates for the blending (Figure 7b). They are rendered once for each adjacent texture using appropriate alpha values at the vertices to gain correct blending. The assignment of alpha values for each vertex for each image is as follows. For each boundary vertex it is decided in which image it is best represented. For the best image the vertex is assigned an alpha value of one, while for all other images it is set to zero. For all surrounding vertices, that are not boundary vertices the alpha value is set to one if the vertex belongs to a triangle that was previously assigned to the current texture (Figure 7c).

Rendering the textured triangles with these alpha values results in a smooth transition. Unfortunately, the blending takes place across the width of only one triangle. If the object is finely tessellated the blending area will become rather small and contrasting textures are still not sufficiently separated. This problem can be solved by computing the blending on an object with coarser tessellation and assigning interpolated alpha values to the vertices of the fine subdivided mesh.

8. Multiple View Registration

When the texture is combined from multiple views a slightly misaligned image can produce visible artifacts since image features blended between two images may not be aligned. The circumstances which can lead to misalignment when only one view is considered are mentioned in Section 5. If we have multiple already registered views an additional similarity measurement s_{tex} can be defined which does not compare silhouettes but the texture of one view to the texture obtained by another view. This results in a global optimization taking into account all views.

8.1 Texture Comparison

Given the parameters (π_1, f_1) and (π_2, f_2) of two registered views and the sets T_1 and T_2 of valid triangles, the quality of the registration can be measured by comparing the textures mapped, one in turn, onto the set of overlapping triangles $T_1 \cap T_2$. The triangles are rendered from the view specified by the averaged parameters $\left(\frac{\pi_1+\pi_2}{2}, \frac{f_1+f_2}{2}\right)$. Choosing the averaged view yields similar loss of quality due to distortion and resampling in both textures.

In the case of a perfectly diffuse surface the textures mapped onto $T_1 \cap T_2$ will look identically, whereas specularity leads to view-dependent highlights which occur in different location on the surface for different views. To get less view-dependent textures the color images are transformed into the HSV color space which separates the brightness (value) from the hue and the saturations. Only the hue and/or saturation-channel are used for comparison avoiding the influence of view-dependent brightness. Of course, also other methods can be applied to create view-independent textures like the one presented in [12], but they tend to be more expensive. However, the hue channel of the two textures can now be compared like the intensity values of two different blurred silhouettes in Section 5. At first, the positive difference of the first texture minus the second texture is rendered into the red channel of the frame buffer and then the reversed difference is rendered into the green channel. Summing up the histogram weighted by the difference values yields a value that becomes minimal when the two views are perfectly aligned. This measure $s_{tex}(\pi_1, f_1, \pi_2, f_2)$ allows to register multiple views with respect to each other.

8.2 Iterative Global Optimization

A registration of multiple views starts with the separate registration of each view based on the silhouette as described in Section 6. After the single-view registration the sets of valid triangles are determined and texture coordinates are computed for the vertices. For each pair of views (i, j) the set of overlapping triangles $T_i \cap T_j$ is determined and the averaged parameters (π_{ij}, f_{ij}) are calculated. For these pairs an initial measurement $s_{ij} = s_{tex}(\pi_i, f_i, \pi_j, f_j)$ is evaluated.

Successively each view i is selected and the set of other views V_i is determined which are sharing overlapping triangles with i. We can now optimize the following function:

$$s_{\text{multiview}}(\pi_i, f_i) = \sum_{j \in V_i} \frac{s_{tex}(\pi_i, f_i, \pi_j, f_j)}{s_{ij}} \qquad (4)$$

Again, the extended downhill simplex method presented in Section 6 can be applied, this time calculating new texture coordinates and evaluating $s_{\text{multiview}}(\pi_i, f_i)$ for each try. Since the changes in π_i are expected to be rather small a simplex with small radius is constructed around π_i and the optimization is already stopped after a few evaluations of $s_{multiview}$. Iterating this process several times over all views until no further updates are performed will produce the best possible registration regarding the surface textures.

9. Results

The presented methods were applied to two different objects, a bird and a moose. The models have been acquired using a Steinbichler Tricolite 3D scanner. The bird's model consists of around 7000 triangles while the moose is tessellated more finely with nearly 11000 triangles. The images were taken with a Kodak DCS 560 digital camera that yields an image resolution of 3040x2008 pixels which we reduced to 1024x676 since the applied graphics hardware cannot deal with larger textures. We run the optimization on a SGI Octane equipped with a MXE graphic board containing 8MB of texture ram.

In Figure 8 the results after automatic registration and stitching of several images onto the models are shown and compared to real photos that have not been used for generating the texture. The moose texture consist of 15 different images taken with two different lenses and at different object distances. The bird was textured using just 10 images.

The synthetic results compare really well to the photos although two kinds of artifacts are visible. At

mode	x·y	value	t_x	t_y	t_z	ϕ_x	ϕ_y	ϕ_z	times
XOR	500x330	avg	7.47554	-5.51689	704.69	-118.956	-43.5465	-119.326	40
		var	0.00589416	0.00646141	1.97692	0.466394	0.0864301	0.282632	
XOR	1000x660	avg	7.55476	-5.55964	706.194	-119.32	-43.5189	-119.719	130
		var	0.00341098	0.00299123	2.41879	0.215139	0.0937607	0.154695	
blurred	500x330	avg	7.26057	-5.64407	706.565	-117.479	-43.0215	-118.237	39
		var	0.000732457	0.00421928	0.165303	0.0381274	0.0163896	0.0305439	
blurred	1000x660	avg	7.3034	-5.67636	706.661	-117.667	-43.0383	-118.386	104
		var	0.0041396	0.000900759	0.301037	0.158217	0.0676453	0.150163	

Table 1. Average value and variance value of the recovered camera parameters and the required time applying the XORed and blurred silhouette matching algorithm for different resolutions. The optimization has been started several times from different positions.

	image proc.	start pos.	opt.	FoV	stitching	total
10 images	235	826	239	365	12	1677
average	23.5	82.6	23.9	36.5	1.2	155
15 images	359	1660	536	1250	21	3826
average	23.9	110.6	35.7	83.3	1.4	255

Table 2. Registration timings (in seconds) for the bird (top rows) and the moose (bottom rows).

the top of the antler some triangles are not textured because they are too close to the outline in each incorporated image. Here, no reliable information could be retrieved. The other artifacts are due to the non-diffuse surface reflectance. Even though the position of the lights was not changed during the acquisition, specular highlights result in brightness differences among the acquired images as can clearly be seen in Figure 9a. To further reduce these lighting artifacts a purely diffuse texture would have to be computed incorporating samples from all acquired pictures.

The precision of the presented algorithm is visualized in Figure 9a where the right front wheel of the moose is shown. The wheel is actually textured by at least six different images. Although the texture of the wheel is composed using several different views, the fine lines of the wood's structure is completely preserved, indicating a very accurate registration.

When comparing the XOR and blurred matching methods, it can be seen from Table 1 that the blurred silhouette method leads to superior results. The variance of the recovered parameters is generally decreased, often by one order of magnitude. From our experiments, it could also be observed that although the computation of the similarity function is computationally more expensive, the optimization converges more quickly for non-ideal starting points.

Table 2 lists the time (in seconds) needed for the registration task of the bird and moose models. The registration of the bird took around 28 minutes, while the moose took 64 minutes since more texture information and a more complex geometric model were used and the resolution used for the final optimization was increased (bird: 500x300, moose: 800x528). The images are first loaded and processed to extract the silhouettes, then a starting point for the optimization is generated, the optimization is run for recovering the position and orientation, the field of view is determined, and finally the textures are stitched onto the model. Most of the time is spent for finding an appropriate starting position and for determining the field of view.

Time could be saved by manually selecting a good starting position. But it turned out that the optimization of the pose and orientation after manual alignment consumed more time (around one minute) since the starting point for the optimization is not as precise as the automatic method. By fixing the field of view during the acquisition further time could be saved, since in that case the field of view had to be determined only once.

All the results presented so far have been computed without using the texture-based multi-view optimization (see Section 8). It turned out that the purely geometry-based registration already produces results of very high accuracy, so that the texture-based matching is only helpful if one of the input images is misaligned for some reason. In our tests, it produces comparable results to those shown here, consuming additional time.

10. Conclusions

We have described a number of novel techniques to register and to stitch 2D images onto 3D geometric models. The camera transformation for each image is determined by an optimization based on silhouette comparison. If the resulting alignment is not accurate enough, further optimization based on texture information is possible. Using the recovered camera transformation, the image is stitched onto the surface. Finally, for multiple views, an algorithm is presented that produces smooth transitions between textures assigned to adjacent surface regions on the model.

The presented methods do not require any user interaction during the entire processing. They work efficiently, exploit graphic hardware features and result in very accurately aligned textures. Differences in the brightness due to specularity are still visible. To further improve the quality of the results, the reflectional properties of the surfaces must also be considered or the algorithm must blend between the textured depending on the selected view-point.

Acknowledgments

Part of this work was funded by the DFG (Deutsche Forschungsgemeinschaft). Thanks to Hartmut Schirmacher and Jan Kautz for proofreading and commenting on this paper, and to Kolja Kaehler, Michael Gösele and Jan Uschok for acquiring the model of the moose and some of the images.

References

[1] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, Dec 1992.

[2] L. Brunie, S. Lavallée, and R. Szeliski. Using force fields derived from 3D distance maps for inferring the attitude of a 3D rigid object. In *Proceedings of Computer Vision (ECCV '92)*.

[3] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH 96*, pages 11–20, August 1996.

[4] P. E. Debevec, Y. Yu, and G. D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. *Eurographics Rendering Workshop 1998*, pages 105–116, June 1998.

[5] B. Guenter, C. Grimm, D. Wood, H. Malvar, and F. Pighin. Making faces. *Proceedings of SIGGRAPH 98*, pages 55–66, July 1998.

[6] H. H. S. Ip and L. Yin. Constructing a 3d individualized head model from two orthogonal views. *The Visual Computer*, 12(5):254–268, 1996.

[7] D. J. Kriegman, B. Vijayakumar, and J. Ponce. Constraints for recognizing and locating curved 3D objects from monocular image features. In *Proceedings of Computer Vision (ECCV '92)*, volume 588 of *LNCS*, pages 829–833. Springer, mai 1992.

[8] D. G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, 1991.

[9] S. R. Marshner. *Inverse rendering for computer graphics*. PhD thesis, Cornelle University, 1998.

[10] K. Matsushita and T. Kaneko. Efficient and handy texture mapping on 3d surfaces. *Computer Graphics Forum*, 18(3):349–358, September 1999.

[11] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *Proceedings of SIGGRAPH 95*, pages 191–198, August 1995.

[12] P. J. Neugebauer and K. Klein. Texturing 3d models of real world objects from multiple unregistered photographic views. *Computer Graphics Forum*, 18(3):245–256, September 1999.

[13] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. H. Salesin. Synthesizing realistic facial expressions from photographs. In *Proceedings of SIGGRAPH 98*, pages 75–84, July 1998.

[14] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C : the art of scientific computing*. Cambridge Univ. Press, 2nd ed. edition, 1994.

[15] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Eurographics Rendering Workshop 1997*, pages 23–34. Springer Wien, June 1997.

[16] C. Rocchini, P. Cignoni, and C. Montani. Multiple textures stitching and blending on 3d objects. In *Eurographics Rendering Workshop 1999*. Eurographics, June 1999.

[17] Y. Sato, M. D. Wheeler, and K. Ikeuchi. Object shape and reflectance modeling from observation. In *Proceedings of SIGGRAPH 97*, pages 379–388, August 1997.

[18] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast shadow and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):249–252, July 1992.

[19] W. Stuerzlinger. Imaging all visible surfaces. In *Graphics Interface '99*, pages 115–122, June 1999.

[20] R. Tsai. A versatile camera calibration technique for high accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics nd Automation*, 3(4), Aug 1987.

[21] Y. Yu and J. Malik. Recovering photometric properties of architectural scenes from photographs. In *Proceedings of SIGGRAPH 98*, pages 207–218, July 1998.

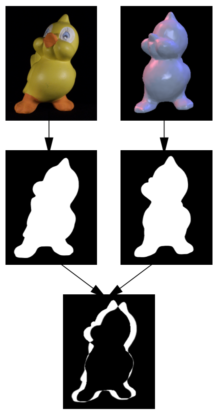

Figure 5. Measuring the difference between the photo (left) and one view of the model (right) by the area occupied by the XOR-ed foreground pixels.

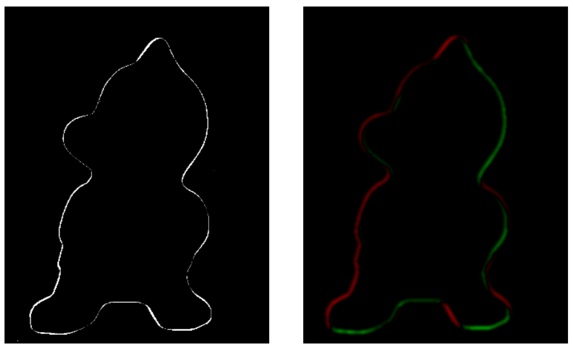

Figure 6. XORed sharp silhouettes (left) and subtracted blurred silhouettes (right).

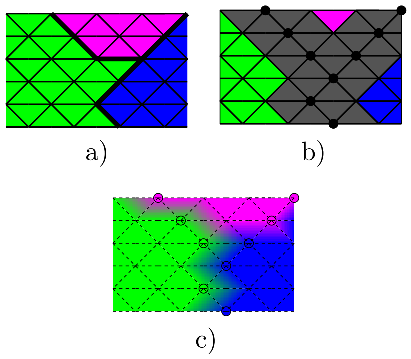

Figure 7. a) Adjacent triangles textured using different images. b) Possibly blended triangles shaded grey. c) Each boundary vertex is assigned to one image and textures are blended.

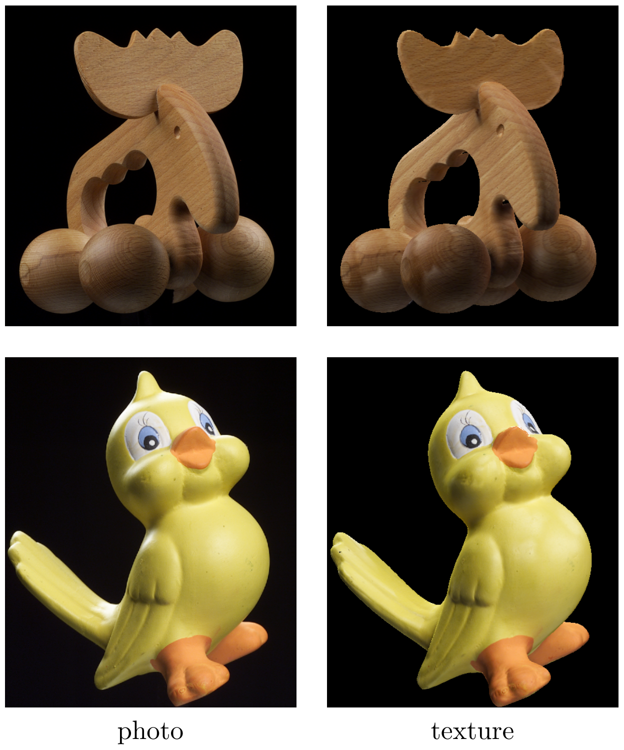

Figure 8. Novel viewpoint. Left column: photo that has not been used to generate the texture. Right column: synthetic model rendered with the generated textures.

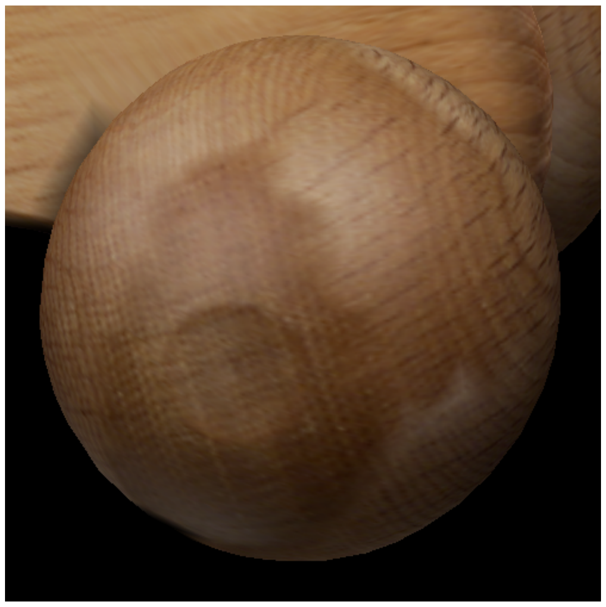

Figure 9. Texture Alignment. View at the right front wheel. Several textures are so accurately aligned that even fine lines in the wood's structure are preserved.