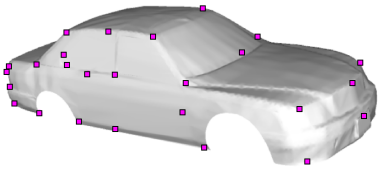
FlexiStickers – Photogrammetric Texture Mapping using Casual Images

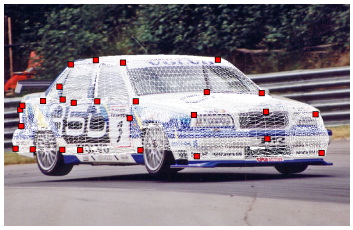
Yochay Tzur*
Technion – Israel Institute of Technology

Ayellet Tal†
Technion – Israel Institute of Technology



(a) A model of a car (18K faces)

(b) A (casual) image of a racing car



(c) Mapping

(d) Result

(e) Result from a different viewpoint

Figure 1: *Texturing a model from a casual image. (a) The input model; (b) A source image; (c) The mapping calculated using 26 constraints; (d-e) The result obtained by our method. In this example, constrained parameterization techniques are unsuitable due to the viewpoint from which the source image was taken. Moreover, the photogrammetric approach is unsuitable due to the difference in shape between the model and the photographed object. Our method is capable of accounting both for the photography effects and for the difference in shape.*

Abstract

Texturing 3D models using casual images has gained importance in the last decade, with the advent of huge databases of images. We present a novel approach for performing this task, which manages to account for the 3D geometry of the photographed object. Our method overcomes the limitation of both the constrained-parameterization approach, which does not account for the photography effects, and the photogrammetric approach, which cannot handle arbitrary images. The key idea of our algorithm is to formulate the mapping estimation as a Moving-Least-Squares problem for recovering local camera parameters at each vertex. The algorithm is realized in a *FlexiStickers* application, which enables fast interactive texture mapping using a small number of constraints.

1 Introduction

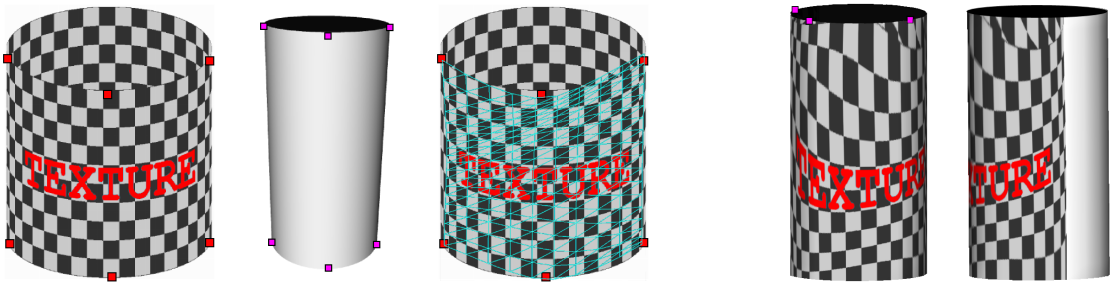
Texture mapping has been a fundamental problem in computer graphics from its early days. As online image databases have become increasingly accessible, the ability to texture 3D models using casual images has gained importance. This will facilitate, for example, the task of texturing models of an animal using any of the hundreds of images of this animal found on the Internet, or enabling a naive user to create personal avatars using the user's own images.

To texture a model using an image, a mapping from the surface to the image should be calculated. Given user-defined constraints, a common approach to establish this mapping is employing *constrained parameterization* [Kraevoy et al. 2003; Lee et al. 2008; Lévy 2001; Zhou et al. 2005]. This approach computes the mapping by embedding the mesh onto the image plane, while attempting to satisfy the constraints and minimize a specific distortion metric. This approach is suitable for casual images, since no prior assumptions regarding the source image and the camera are made. However, inherent distortions might be introduced due to photography effects that result from the viewpoint and the object's 3D geometry.

This problem is demonstrated in Figure 1. We are given a casual image taken from a viewpoint that enhances the 3D geometry of the photographed car (i.e., there exists a large difference in the visible depth values). We are also given a disc-like model of a car, whose unconstrained parameterization (e.g., minimizing angles) is very good. Constrained parameterization, even when using a large number of constraints, cannot produce a satisfactory mapping, since its two goals – minimizing distortions and satisfying constraints – conflict. In our experiment, even after assigning 40 points, the results were still unsatisfactory. This is due to the large displacement between the photograph and the parameterization.

If the model and the photographed car were highly similar, a *photogrammetric* approach could solve this problem, by recovering the camera's parameters. Using these parameters to reproject the model onto the source image, would compensate for the photography dis-

*e-mail: ytzur@cs.technion.ac.il
†e-mail: ayellet@ee.technion.ac.il



(a) Source image (b) Model (c) Parameterization (d) Texture mapping using constrained parameterization

Figure 2: *The inherent drawback of the constrained parameterization approach. (a) A "photographed" cylinder is used as a source image, in which the text appears curved and the squares in the center seem wider than those near the silhouettes. These photography effects result from the viewpoint and the object's 3D geometry. (b) The model is a cylinder with different proportions. Six constraints are specified on the image (red) and on the model (magenta). (c) The result of constrained parameterization strives to minimize the parameterization distortions, ignoring the photography effects. (Here, we use the publicly available code of [Lévy 2001].) (d) The resulting texture mapping is incorrect – the text remains curved and the squares differ in size. Even with a large number of constraints, we could not achieve satisfactory results.*

tortions [Debevec et al. 1996; Weinhaus and Devich 1999]. But, in this example, the model and the photographed object are dissimilar.

The major challenge is hence to handle the photography effects as well as the differences in pose and proportion between the photographed object and the model. This paper proposes a novel algorithm that achieves this goal. The key idea is to recover the camera parameters, as done in the photogrammetric approach. However, since no global parameters exist, this is done locally, for each vertex of the mesh independently. An important consideration in this scheme is how to properly weight the user's constraints at each vertex. We formulate this parameter estimation problem as a Moving-Least-Squares problem, whose solution is demonstrated to account for the photography effects. For instance, the model of Figure 1 was textured using only 26 constraints, yielding a satisfactory result.

It is important to note that our technique does not perform parameterization, but rather projection of the model according to the estimated local cameras. Therefore, issues such as foldovers are not a concern, since the visibility algorithm will address these.

The contributions of this paper are threefold:

- A new algorithm is proposed for texturing 3D models from casual images (Sections 3–4). The algorithm provides the advantages of the photogrammetric approach and the flexibility of the constrained-parameterization approach.
- Our algorithm is realized in an interactive system that enables fast texturing of 3D models, utilizing a small number of features, typically 10–30 (Section 5).
- We introduce a new visibility detection technique for finding the regions of the model that should be textured.

2 Related Work

Texture mapping using a casual photograph is usually performed using *constrained parameterization* [Hormann et al. 2007]. The model–image correspondence is calculated by unwrapping the manifold onto the image, constraining the parameterization by user-defined features. Low-distortion parameterizations attempt to minimize distortion using a variety of considerations, such as angular distortion, Dirichlet energy, and edge length. Examples of constrained-parameterization techniques are [Lévy 2001; Desbrun et al. 2002; Kraevoy et al. 2003; Zhou et al. 2005; Schmidt et al. 2006; Gingold et al. 2006; Lee et al. 2008; Tai et al. 2008].

While producing pretty results, visual distortions may occur in the textured model either when the photographed object exhibits high variance in depth or when the source image is taken from a specific viewpoint. This is so, since constrained parameterization relies entirely on the model's geometry, ignoring the photography effects, as illustrated in Figure 2. Sometimes, the user can manually compensate for these effects, by providing a large number of constraints. However, this will not always solve the problem.

An alternative approach, which takes into account the 3D geometry of the photographed image, is *photogrammetric* [Debevec et al. 1996; Weinhaus and D. 1997; Weinhaus and Devich 1999; Lensch et al. 2000; Bernardini et al. 2001; Colombo et al. 2005; Sinha et al. 2008; Xiao et al. 2008]. In this approach, it is assumed that the photographed object is highly similar to the model to be textured and that it was acquired using a camera of a known model (e.g., a *pinhole camera*). The missing camera parameters are estimated and the recovered camera is used to reproject the model onto the source image, implicitly defining the model–image mapping.

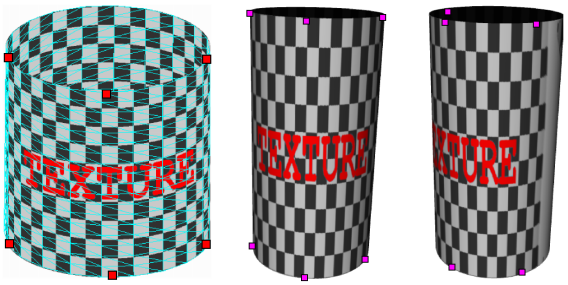
The major advantage of the photogrammetric approach is that it compensates for the photography effects introduced by the camera projection. However, the inherent limitation of the approach is the requirement that the photographed object and the model are similar, prohibiting the use of casual images. An additional disadvantage of some of these methods is the need for added information regarding the camera (e.g., calibration, position and orientation) or the model. Therefore, texturing performed using this approach usually utilizes images especially photographed for this purpose, where the 3D model is identical to the photographed object. Often, reconstruction and texture mapping are jointly performed.

Our goal is to enable texture mapping using casual images, as commonly done by constrained parameterization, while accounting for the photography effects, as achieved by the photogrammetric methods. This will not only ease the task of texture mapping, but will make it possible in cases that cannot be achieved otherwise.

3 Algorithm Outline

We are given a mesh $\mathcal{M}$, a texture source image I_s, and two sets of corresponding user-defined features (*constraints*) $\{p_i \in \mathcal{M}\}_{i=1}^{n}$ and $\{q_i \in I_s\}_{i=1}^{n}$. We are seeking a mapping $T : \mathcal{M} \to I_s$, s.t. $T(p_i) = q_i$, $\forall i, 1 \leq i \leq n$, which will compensate for the photography effects presented in I_s, as demonstrated in Figure 3.

We begin by reviewing the general photogrammetric approach, in



(a) Mapping (b) Textured model

Figure 3: *Our result for the synthetic example in Figure 2. (a) The mapping compensates for the photography effects. (b) In the result, the text is parallel to the edges of the cylinder and has equal-width characters, as expected. Only 6 constraints are used.*

which the desired mapping is a global camera projection T_G ([Hartley and Zisserman 2004]):

$$q = T_G(p) = KR[I|-C]\,p.$$

Here, K is the camera calibration matrix (the camera's *internal parameters*), R is a rotation matrix, and C is a translation vector. Together, they describe the relative orientation and position of the camera with respect to the photographed object (the camera's *external parameters*). If the model fully conforms to the photographed object, there exists a set of internal and external parameters that satisfy the constraints. A common way to estimate these parameters is to minimize the reprojection error of the features, given by:

$$E(K,R,C) = \sum_{i=1}^{n} \|KR[I|-C]p_i - q_i\|^2.$$

Since we are using a casual image, it is likely that the photographed object and the model will substantially differ in shape. In this case, the resulting T_G will produce an incorrect mapping, both in projecting the features themselves and in terms of the texturing quality.

Our algorithm is inspired by the photogrammetric approach. However, instead of estimating a single global camera for the entire model, our key idea is to find a local camera projection $T^{(v)}$ for each vertex v. This is done by weighting the constraints non-uniformly according to their distance from v. Once $T^{(v)}$ is recovered for every vertex v, $T(p)$ is defined for every internal point $p \in \mathscr{M}$, using its barycentric coordinates relative to its adjacent vertices. Section 4 describes how to derive transformations $T^{(v)}$ in two important cases: the similarity case and the affine case.

The use of a camera projection introduces a visibility problem: It usually maps at least two mesh points to each image position (the model's front and back), from which only one should be textured, while the others should be considered invisible. In contrast, note that an inherent property of the constrained parameterization approach is that at most one mesh point is mapped to each image point. Therefore, in order to texture the model properly, we should determine the visible mesh points. In the general photogrammetric approach, this problem is addressed using visible surface detection (e.g., by a Z-Buffer or by ray-tracing). Unfortunately these methods are not applicable in our case since the global model's depth is unknown.

Section 5 presents our *D-Buffer algorithm*, which addresses this problem. The intuition behind this algorithm is that the constraints defined by the user give us sufficient information regarding the portions of the mesh that should be textured.

Our algorithm textures the model in four steps. First, the model is decomposed into a texture atlas [Lévy et al. 2002; Zhang et al. 2005]; In our implementation, we use [Lévy et al. 2002]. This produces a blank atlas which is colored in the next steps. Second, the model-image mapping is computed. Third, using the mapping and the user constraints, the regions of the model that are visible in the source image are detected. Finally, the atlas pixels that correspond to the model's visible regions are textured, based on the mapping. Our contributions are algorithms for performing the tasks of Steps 2 and 3. They are presented in Sections 4 and 5, respectively.

4 Mapping Recovery

This section describes the derivation of the local camera projection at vertex v:

$$T^{(v)}(p) = K^{(v)}R^{(v)}[I|-C^{(v)}]p.$$

For each v, a different transformation is derived. This is achieved by weighting the constraints differently at each vertex, in contrast to the global case. This requirement can be formulated by the following minimization problem of the feature reprojection error:

$$E^{(v)}(K^{(v)},R^{(v)},C^{(v)}) = \sum_{i=1}^{n} {w_i}^{(v)} \|K^{(v)}R^{(v)}[I|-C^{(v)}]p_i - q_i\|^2. \quad (1)$$

Here, ${w_i}^{(v)}$ is the weight of the p_i-q_i constraint in the estimation of $T^{(v)}$. The question is how to assign the weights, so as to take into account the expected influence of each constraint on vertex v.

There are several possible weighting schemes. We define the weights according to their proximity, i.e., if p_i is closer to v, it will have a larger impact on the estimation than constraints defined in farther points. Let $D(\cdot,\cdot)$ be the geodesic distance between two points on the mesh surface. Our weights are defined as:

$${w_i}^{(v)} = \frac{1}{\alpha + D(v,p_i)^{\beta}}. \quad (2)$$

In our implementation, $\alpha = 10^{-3}$ and $\beta = 2$. Note that this weighting ensures that $T^{(p_i)}(p_i) = q_i \;\forall 1 \le i \le n$, as requested from T.

In order to find $T^{(v)}$ that minimizes Equation (1), we assume a simple camera model, in which the projection is a *weak perspective*, i.e. $d_{max} - d_{min} \ll d_{min}$, where d_{min} and d_{max} are the distances of the closest and farthest object points from the camera. We also assume that the camera has no skew. These assumptions usually conform to casual images, since most modern digital cameras have no skew, and the objects are often photographed from distance.

Under this camera model, the mapping can be rewritten as:

$$T^{(v)}(p) = M^{(v)}p + c^{(v)}, \quad (3)$$

with

$$M^{(v)} \in \mathbb{R}^{2\times 3} = \begin{pmatrix} {m_1}^{(v)T} \\ {m_2}^{(v)T} \end{pmatrix}$$

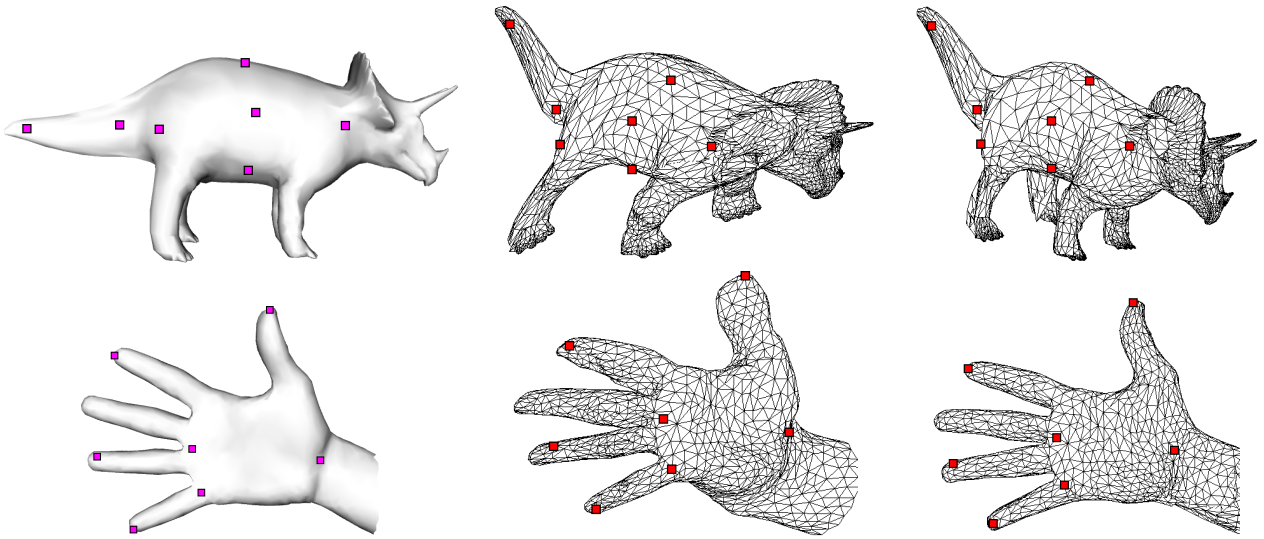
being the world-to-image projection and $c^{(v)} \in \mathbb{R}^2$ is the translation in the image plane.

Moreover, $R^{(v)}$ is orthonormal, thus $M^{(v)}{M^{(v)}}^T$ is diagonal, or

$${m_1}^{(v)T} {m_2}^{(v)} = 0.$$

Finally, assuming also that the camera has a uniform scaling in both axes, we get an additional constraint:

$${m_1}^{(v)T} {m_1}^{(v)} = {m_2}^{(v)T} {m_2}^{(v)}.$$



(a) Models with feature points (b) Applying the affine transform (c) Applying the similarity transform

Figure 4: *Comparison of the affine solution and the similarity solution. The user specified 7 feature points on the model (magenta) and their desired locations on the image plane (red). Using these constraints, both the affine solution and the similarity solution are computed and applied to the model. Figures (b)-(c) show the results. While the constraints are mapped correctly in both cases, it can be seen that the affine solution results in visible distortions, which are avoided in the similarity mapping.*

Putting it all together, our optimization problem at each mesh vertex attempts to minimize the error $E^{(v)}(M^{(v)}, c^{(v)})$:

$$\min_{\{M^{(v)},c^{(v)}\}} \sum_{i=1}^{n} w_i{}^{(v)} \|M^{(v)} p_i + c^{(v)} - q_i\|^2 \tag{4}$$

subject to:

$$\begin{aligned} m_1{}^{(v)T} m_2{}^{(v)} &= 0, \\ m_1{}^{(v)T} m_1{}^{(v)} &= m_2{}^{(v)T} m_2{}^{(v)}. \end{aligned}$$

In this optimization problem, the transformation sought is a similarity. We derive its solution in Section 4.2. We also solve this problem for an easier special case – the affine case – which often produces sufficiently appealing results, with less computational effort (Section 4.1). Note that [Schaefer et al. 2006] define a similar optimization for warping 2D images. Our problem and hence our solution are different, since we are addressing the 3D-2D case.

4.1 Solution to the Affine Case

In the affine case, Equation (4) becomes an unconstrained optimization problem, which can be solved by differentiation. Equation (4) is differentiated with respect to $c^{(v)}$ and compared to 0, yielding:

$$c^{(v)} = q_*{}^{(v)} - M^{(v)} p_*{}^{(v)},$$

where $q_*{}^{(v)}, p_*{}^{(v)}$ are the weighted constraint centroids:

$$q_*{}^{(v)} = \frac{\sum_{i=1}^{n} w_i{}^{(v)} q_i}{\sum_{i=1}^{n} w_i{}^{(v)}} \;\;,\;\; p_*{}^{(v)} = \frac{\sum_{i=1}^{n} w_i{}^{(v)} p_i}{\sum_{i=1}^{n} w_i{}^{(v)}}.$$

Substituting $c^{(v)}$ into Equation (3) gives the following simplified expression, in which $c^{(v)}$ is eliminated:

$$T^{(v)}(p) = M^{(v)}(p - p_*{}^{(v)}) + q_*{}^{(v)}.$$

Thus, the reprojection error (Equation (4)) can be rewritten with respect to the weighted centroids as:

$$E^{(v)} = \sum_{i=1}^{n} w_i{}^{(v)} \|M^{(v)} \hat{p}_i{}^{(v)} - \hat{q}_i{}^{(v)}\|^2, \tag{5}$$

where

$$\hat{p}_i{}^{(v)} = p_i - p_*{}^{(v)} \;\;,\;\; \hat{q}_i{}^{(v)} = q_i - q_*{}^{(v)}.$$

This reduces to the classical weighted-least-squares problem, having a closed solution, given by the *Normal Equations*:

$$M^{(v)} = \left(\sum_{j=1}^{n} w_j{}^{(v)} \hat{q}_j{}^{(v)} \hat{p}_j{}^{(v)T} \right) \cdot \left(\sum_{i=1}^{n} w_i{}^{(v)} \hat{p}_i{}^{(v)} \hat{p}_i{}^{(v)T} \right)^{-1}.$$

Matrix $M^{(v)}$ and vector $c^{(v)}$ are the desired solution for vertex v.

4.2 Solution to the Similarity Case

The affine solution produces satisfying results in most cases. However, in some cases, usually characterized by more complex articulation, shape and a small number of constraints, the angle and edge-length distortions are visible. The similarity solution solves these problems, as illustrated in Figure 4. In this figure, starting from the same constraints, the affine solution exhibits high distortions, in comparison to the similarity solution.

The similarity case presented in Equation (4), when solved for each vertex v separately, reduces to the problem of calibrating a scaled-orthographic camera per vertex. This problem has been addressed in the photogrammetric literature ([Hartley and Zisserman 2004; Bregler et al. 2004; Brox et al. 2007]). However, solving these calibration problems as a set of independent optimizations, might lead to very different estimations of nearby vertices. This causes high distortions in the resulting mapping. Below, we first present a possible solution to the optimization problem and then describe a technique that overcomes the above drawback.

Solution to the optimization problem: Similarly to the affine case, the solution to the similarity case begins by eliminating $c^{(v)}$, yielding Equation (5).

However, in contrast to the affine case, which has a closed-form solution, finding $M^{(v)}$ in the similarity case is non-linear and is solved using an iterative process. In the following, we omit the $^{(v)}$ notation, keeping in mind that the procedure is performed for each vertex v independently.

From the constraints of Equation (4), it is clear that $M = sKR$, where s is scalar ($s = \|m_1\| = \|m_2\|$), $R \in SO(3)$, and

$$K = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Thus, Equation (5) can be rewritten as

$$\min_{s,R} \sum_{i=1}^{n} w_i \|r_i\|^2, \tag{6}$$

with

$$r_i = \hat{q}_i - sKR\hat{p}_i. \tag{7}$$

Equation (6) has four degrees of freedom: s and three degrees of freedom in R. We solve it using the *axis-angle* representation of R:

$$R = e^{\hat{\omega}\theta}, \tag{8}$$

where $\omega \in \mathbb{R}^3$ is the 3D rotation axis of R, θ is the angle of rotation around ω, and $\hat{\omega}$ is the skew-symmetric matrix defined by ω:

$$\hat{\omega} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}.$$

From this representation, R can be efficiently calculated using the Rodrigues' rotation formula [Murray et al. 1994]:

$$R = e^{\hat{\omega}\theta} = I + \hat{\omega}\sin\theta + \hat{\omega}^2(1-\cos\theta).$$

Equation (6) is linearized using the Taylor expansion of Equation (8):

$$e^{\hat{\omega}} = I + \hat{\omega} + \frac{\hat{\omega}^2}{2!} + \frac{\hat{\omega}^3}{3!} + \ldots \approx I + \hat{\omega}. \tag{9}$$

Substituting Equation (9) into Equation (6) yields a linear weighted least squares problem (with $\nu = s\omega$):

$$\min_{s,\nu} \sum_{i=1}^{n} w_i \|\hat{q}_i - \begin{pmatrix} s & -\nu_z & \nu_y \\ \nu_z & s & -\nu_x \end{pmatrix} \hat{p}_i\|^2.$$

As before, the Normal Equations are used to find the minimum:

$$\vec{x} = (A^T W A)^{-1}(A^T W B),$$

with $\vec{x} = (s \;\; \nu_x \;\; \nu_y \;\; \nu_z)^T$, and A, B, W are defined as:

$$A = \begin{pmatrix} \hat{p}_{1,x} & 0 & \hat{p}_{1,z} & -\hat{p}_{1,y} \\ \hat{p}_{1,y} & -\hat{p}_{1,z} & 0 & \hat{p}_{1,x} \\ \hat{p}_{2,x} & 0 & \hat{p}_{2,z} & -\hat{p}_{2,y} \\ \hat{p}_{2,y} & -\hat{p}_{2,z} & 0 & \hat{p}_{2,x} \\ \cdots & \cdots & \cdots & \cdots \\ \hat{p}_{n,x} & 0 & \hat{p}_{n,z} & -\hat{p}_{n,y} \\ \hat{p}_{n,y} & -\hat{p}_{n,z} & 0 & \hat{p}_{n,x} \end{pmatrix}, B = \begin{pmatrix} \hat{q}_{1,x} \\ \hat{q}_{1,y} \\ \hat{q}_{2,x} \\ \hat{q}_{2,y} \\ \cdots \\ \hat{q}_{n,x} \\ \hat{q}_{n,y} \end{pmatrix},$$

$$W = diag(w_1, w_1, w_2, w_2, \ldots, w_n, w_n).$$

Since the Taylor expansion is used, the linear approximation in Equation (9) is best utilized when R represents a small rotation. Therefore, the above method is used iteratively to estimate the relative rotation between consecutive iterations, rather than using the absolute rotation. Below we describe this iterative process.

Let s_t and R_t be the estimations for s and R at iteration t and s_{t+1}, R_{t+1} be the estimations to s and R at the next iteration. Define:

$$\begin{aligned} s_{t+1} &= (1+\Delta s_t)s_t, \\ R_{t+1} &= \Delta R_t \cdot R_t, \end{aligned} \tag{10}$$

where Δs_t and ΔR_t are the relative scaling and rotation, respectively. Substituting these into Equation (7) yields:

$$r_i = \hat{q}_i - s_{t+1}KR_{t+1}\hat{p}_i = \hat{q}_i - K[(1+\Delta s_t)\Delta R_t]s_t R_t \hat{p}_i.$$

Defining the transformed constraints at iteration t, $p_{i,t} = s_t R_t \hat{p}_i$, matrix A is constructed from $\{p_{i,t}\}$ rather than from $\{\hat{p}_i\}$. This A is used for finding $\vec{x}$ – the estimation to the relative scaling Δs_t and rotation ΔR_t. Finally, we employ Equation (10) to accumulate R and S. This is repeated until the relative rotation and scale between consecutive iterations are smaller than a pre-defined threshold. In practice, it converges after 5-6 iterations.

Ordering the vertex-optimization problems: As explained above, solving the optimization problem for each vertex independently might result in visible distortions. In order to address this problem, we propose to process the vertices in a certain order, and initialize the iterative process described above (for each vertex), using the solutions obtained for previously-processed vertices. While this ordering does not increase the complexity, it greatly improves the quality of the result.

The first processed vertex v_0 is chosen as the vertex whose sum of geodesic distances to the constraints is minimal. Intuitively, this is a "central" vertex in the mapping. Once v_0 is determined, the other vertices are ordered in a Breadth-First-Search (BFS) manner.

To estimate $M^{(v)}$ at a vertex v, we use $M^{(u)}$ as the initial guess, where u is the parent of v in the BFS-tree. With this initialization, the estimations obtained for two adjacent vertices are similar.

To find the initial guess for the seed vertex v_0, we start from the affine solution for v_0 (Section 4.1). In particular, let $M_{aff}^{(v_0)}$ be this affine solution and let $M_{aff}^{(v_0)} = UDV^T$ be its Singular Value Decomposition. Here, $U \in \mathbb{R}^{2\times 2}$ and $V \in \mathbb{R}^{3\times 3}$ are orthonormal matrices (satisfying the constraints of Equation (4)) and D is diagonal:

$$D = \begin{pmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \end{pmatrix}.$$

We define the initial guess of vertex v_0 by:

$$s_1{}^{(v_0)} = (d_1 + d_2)/2, \quad R_1{}^{(v_0)} = UKV^T.$$

Note that in this estimation of the first-iteration rotation, $R_1{}^{(v_0)}$ has only two rows. In order for $R_1{}^{(v_0)}$ to be a rotation, the third row is set to the cross product of the first two rows.

5 Detection of the Visible Regions

When using an image as a texture source, it is essential to determine the model's regions whose corresponding regions in the image are visible, and to texture only them. A major benefit of the

photogrammetric approach for texture mapping is that visible surface detection can be calculated directly from the estimated model-image mapping. Unfortunately, this method is not applicable in our algorithm, as it produces direct mapping from the model to the image plane without estimating the camera's external parameters, hence not providing the depth information. Moreover, even if we were able to estimate the depth, the local per-vertex transformation would result in a non-rigid warp of the model, which might make some back vertices have smaller depth than that of front vertices. This would make some regions classified incorrectly.

Below we describe our *D-Buffer (Distance-Buffer)* algorithm that detects the visible regions. It is based on the observation that the model constraints $\{p_i\}_{i=1}^{n}$ are placed near the surface regions that the user wants to texture – regions that the user considers visible.

For each $q \in I_s$, let $P(q) = \{p \in \mathscr{M} | T(p) = q\}$ be the set of model points that are mapped to q by T. Since only one model point in $P(q)$ is visible, the color information at q should be used to texture only this point.

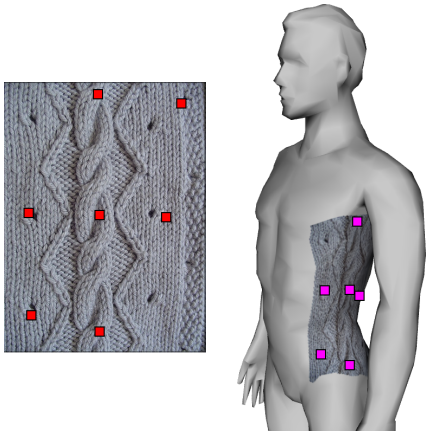
Denote by $D(p, p_i)$ the geodesic distance between $p \in \mathscr{M}$ and a feature point $p_i \in \mathscr{M}$. Let $F(p)$ be the sum of the geodesic distances from p to all the constraints:

$$F(p) = \sum_{i=1}^{n} D(p, p_i).$$

We define $p_{vis}(q) \in P(q)$ – the visible point for q – as the point that has the smallest value of $F(p)$, among all the points in $P(q)$:

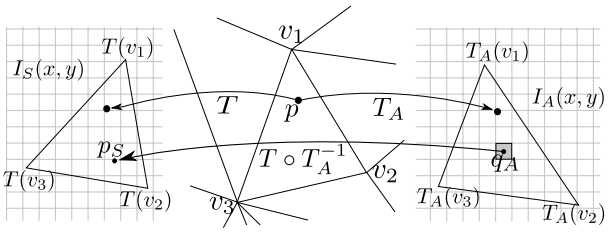
$$p_{vis}(q) = \underset{p \in P(q)}{\operatorname{argmin}} F(p).$$

A desirable property of this visibility definition, illustrated in Figure 5, is that it lets the user texture mesh regions that are occluded in the model under transformation T. In this example, T maps the left arm, the torso, and the right arm to the same image area. While the torso is occluded by the left arm, it is obvious that the user wishes to texture the torso using the image of a woven material. The D-Buffer algorithm picks the correct points, since they are geodesically closer to the feature set.



(a) Source image (b) Texturing result

Figure 5: *Texturing the visible regions. In order to texture the left side of the torso using the image of a woven material, seven constraints are specified. Though the computed T results in occlusion of the torso by the left arm, the D-Buffer algorithm correctly determines the region to be textured.*



(a) Source image (b) Model (c) Texture atlas image

Figure 6: *Painting the atlas from the source image. A mesh triangle (v_1, v_2, v_3) is mapped by T_A to the atlas I_A. The model-image transformation T maps each mesh point p to the texture source I_S. The pixel $q_A \in I_A$ is colored using the color in $I_S(T(T_A^{-1}(q_A)))$.*

The D-Buffer algorithm: The visibility-detection technique described above assumes a continuous representation of the image and the mesh. Realizing it by calculating $p_{vis}(q)$ for every source pixel q would be very inefficient. We describe an algorithm that discretizes and accelerates this computation. The key idea is to rasterize the value of F over the source image and compare it with the value of the corresponding pixel in the atlas. Since this procedure is similar in spirit to the conventional Z-Buffer algorithm, it allows us to implement it efficiently on the hardware, as explained hereafter.

Given a model $\mathscr{M}$, we utilize a texture atlas that represents the model's color information, as commonly done (Figure 6). The atlas consists of a texture image I_A and a piecewise linear mapping $T_A : \mathscr{M} \rightarrow I_A$. The model is textured by assigning a color to every pixel $q_A \in I_A$ from a pixel $p_S \in I_S$, which is found by:

$$p_S = T(T_A^{-1}(q_A)). \tag{11}$$

Hence, the inverse mapping $T_A^{-1}(q_A)$ is applied to find the corresponding mesh point p; the model-image transformation T is used to map p to I_S. Note that we refer only to pixels in I_A that are actually used for texturing.

Obviously, p_S may have multiple atlas pixels mapped to it. The detection of the visible atlas pixels is performed in two steps:

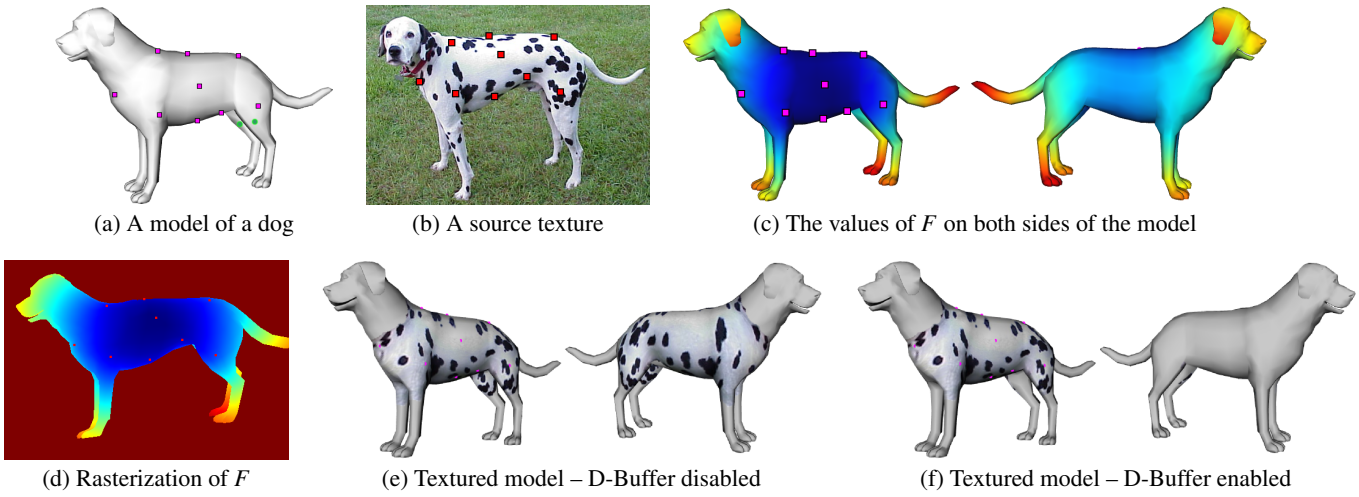
1. Rasterizing the mesh onto I_s using $F(p)$ as the depth value of p (instead of the conventional z coordinate).
2. Mapping each atlas pixel q_A to I_s and comparing the expected F value with the value stored in the depth buffer, to determine its visibility.

In the first step, the mesh triangles are rasterized onto the source image, by mapping the vertices to the image using T, where the depth value of each vertex is its F value. The usual hardware implementation of the Z-Buffer algorithm is used, with the values of F replacing the conventional z coordinates.

Once this rasterization is done, the hardware depth buffer B contains an image of the same size as I_S, in which every pixel q holds the value of the minimal $F(p)$, over all $P(q)$. Thus,

$$B(q) = \min_{p \in P(q)} F(p) = F(p_{vis}(q)).$$

In the second step, each atlas pixel q_A is mapped to $p_S \in I_s$ by applying Equation (11). First, $F(p)$ is calculated for $p = T_A^{-1}(q_A)$, using Barycentric coordinates. Then, the color of $I_S(p_S)$ is copied to q_A only if $F(p) = B(p_S)$.



(a) A model of a dog (b) A source texture (c) The values of F on both sides of the model

(d) Rasterization of F (e) Textured model – D-Buffer disabled (f) Textured model – D-Buffer enabled

Figure 7: *The D-Buffer algorithm. (a) A given model of a dog; (b) A source image. Using 9 user-constraints (magenta & red), the calculated T maps 2-6 model points to each image point. For instance, 4 points on the dog's rear legs (green) are mapped to the same image point. (c) The values of F on both sides of the model, with blue regions correspond to lower values. Notice that F is smaller near the features. (d) The rasterization of F onto the source image domain. (e) The model is textured, disabling the D-Buffer algorithm. Both sides of the model are colored. (f) When the D-Buffer algorithm is enabled, only the visible region of the model is textured, as expected.*

Figure 7 demonstrates this process. A model of a dog is textured using a casual image of a dalmatian, which differ both in shape and in pose. T maps several model points to the same image point, from which only one point should be textured. For instance, four different model points on the rear legs are mapped to the same image point. Figure 7(c) visualizes the value of F, where the blue regions correspond to smaller values of F. Figure 7(d) shows the rasterization of the model onto the image domain using F as the z coordinate. Among the four model points, only the one having the minimal F, is textured. Figure 7(e) shows the result without applying the D-Buffer algorithm, while Figure 7(f) shows our final result. It can be seen that only the regions that relate to the user's features are textured (e.g., the back is not textured).

6 Results

To realize our method, we developed an interactive texturing system called *FlexiStickers*, as illustrated in Figure 8. FlexiStickers enables the user to define a set of "stickers" – mappings from the model to selected image regions – and easily modify them.

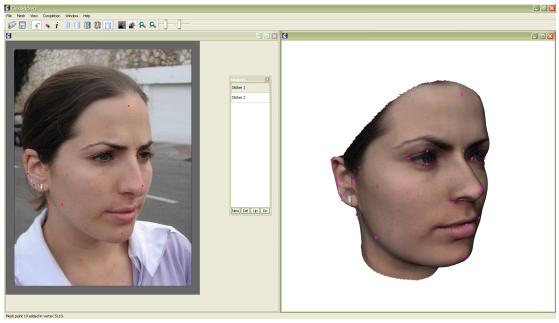


Figure 8: *A screenshot from the FlexiStickers application*

Figure 9 demonstrates the use of FlexiStickers. The user begins by specifying a small number of feature points (at least five) both on the model and on the source image. Given these constraints, our algorithm calculates the mapping and textures the model in interactive time. The user can then improve the result, by either adding more features or adjusting the positions of the existing features. Each modification to the feature set initiates the recalculation of the mapping and the repainting of the atlas.

Figures 10–14 show some of our results – all obtained in 1–10 minutes of user interaction. For Figures 10 and 12, in which the models are relatively simple in terms of articulation and shape, the affine transformation is used. Below we also discuss the use of the alternative approach of constained parameterization for these examples.

Figure 10 shows an example for which constrained parameterization methods work well and require a small number of feature points. Our algorithm manages to produce comparable results, requiring a similar user effort.

Figure 11 presents an example that can be compared to the result obtained by the constrained parameterization algorithm of [Kraevoy et al. 2003]. The model of Igea is textured by an image of a face, using two stickers, one for each side of the head. While the results compare well, our algorithm requires only 30 constraints (for both stickers), whereas [Kraevoy et al. 2003] require almost 100, resulting in greater user effort. This large feature set is mandatory in constrained parameterization, since there exists a very large displacement between the original unconstrained parameterization and the final mapping that suits the image. Moreover, our algorithm need not cut the mesh into two separate discs, thus avoiding the use of extra features for performing the stitching. Instead, a simple blending of the stickers in the overlapping regions suffices; Methods such as [Zhou et al. 2005] can also be utilized.

Figure 12 shows an example for which constrained parameterization does not work well. Here, a model of a coffee mug is textured using a casual image of a mug. The image presents high distortions due to the viewpoint and the 3D geometry of the photographed mug. In particular, the text on the mug appears curved and the letters decrease in size as they get closer to the silhouettes. Our algorithm compensates for these effects and produces an undistorted and realistic result. This is done using only six features and less than a minute of user-time.

First interaction:

(a) Model (b) Source image (c) Mapping based on 5 constraints (d) Result

Second interaction:

(e) Mapping based on 13 constraints (f) Result (g) Atlas

Figure 9: *Interactive texturing of a shoe model. The user starts by positioning five constraints on the model (a) and the image (b). The resulting mapping (c) produces unsatisfying texturing (d), e.g., around the toe. The user adds eight more constraints and relocates some of the constraints (e). The mapping is calculated interactively during the repositioning of the features, and the model is textured accordingly (f). Notice that in the colored atlas (g) only the visible regions are textured. The entire texturing process took 3 minutes of user interaction.*

(a) Model (24K faces) (b) Source image (c) Mapping (d) The textured atlas (e) The resulting textured face

Figure 10: *Texturing a face using an image of a tiger and 19 user-defined constraints. This result is similar to those presented by constrained parameterization methods, e.g., [Kraevoy et al. 2003; Lévy 2001].*

Figure 13 demonstrates a texturing of an entire model of a woman. Two images are used as texture sources, each provides a sticker for one side of the model. The entire mapping is calculated from 75 features. Notice that our algorithm does not distort the text on the woman's shirt, since it accounts for the angle changes that result from the 3D geometry of the photographed woman.

Finally, Figure 14 shows the results of texturing a model using a variety of casual images. Our algorithm succeeds at handling the different proportions and articulations of the photographed men.

Running times: During interaction with the user, two types of calculations take place: mapping and repainting the atlas. The mapping is performed in real-time on a 1.6Ghz Pentium 4-processor machine with 2Gb of memory. For a model of $24K$ faces and 19 constraints (Figure 10), the mapping takes 50 ms. The bottleneck of the process is the atlas repainting, which involves millions of pixels (we use an atlas of 1024×1024 pixels). To overcome this limitation, during the interactive constraint relocation, we use an atlas of half the resolution, performing repainting in 100 ms. Meanwhile, the full atlas is painted in a background thread (typically in 1 second), and presented to the user once the interaction is over.

Limitations: If the photographed object exhibits large articulation compared to the model, the user is required to specify a large number of constraints in each articulated segment. For instance, texturing the model in Figure 14 with the rightmost image, whose articulation differs considerably, required 12 constraints for each arm (comparable to 3-6 constraints used for the other images). One way to handle this problem is to use a weighting function (Equation (2)) that takes dihedral angles into account.

7 Conclusion

This paper has presented a novel method for texturing three-dimensional models using casual images. It provides the advantages of the photogrammetric approach and the flexibility of the constrained-parameterization approach. Since it accounts for the photography effects, it allows us to texture models using source images that cannot be utilized by other approaches.

An additional contribution of this paper is a new visibility detection technique for finding the regions of the model that should be textured, based on the user's constraints. Moreover, it is easily implemented using conventional hardware.

(a) Model (4000K faces) (b) Source image (c) Result (d) Another source image

Figure 11: *Texturing Igea. Our result can be compared to Figure 1 in [Kraevoy et al. 2003] (not included for copyrights reasons). We use only 30 constraints, compared to* ~ 100 *in [Kraevoy et al. 2003]. Moreover, in our case, the original image suffices for texturing, and there is no need to reflect the source image in order to texture both sides of the face.*

(a) Model (11K faces) (b) Source image (c) Mapping (d) Result (e) Result (orthographic)

Figure 12: *Texturing a mug using only six constraints. The result shows how the mapping compensates for the photography effect of the image. In particular, the orthographic view demonstrates that the text is parallel to the edges of the mug, as expected. This result is hard to achieve using the alternative methods.*

Our algorithm is realized in an interactive system that enables fast texturing of 3D models. We also show that only a small number of features are needed to texture the models. The algorithm thus outperforms other texturing methods, both in terms of user effort and in handling cases that are difficult to address otherwise.

In the future, we would like to also consider curve constraints. This may require less user effort in specifying the constraints. Another possible enhancement of the method is automatic "suggestion" of constraints, based on salient points of the model and the image.

Acknowledgements: This research was supported in part by the Israel Science Foundation (ISF) 628/08, S. and N. Grand Research Fund, and the Ollendorff foundation. We thank Shirtmate.com, Tony Harrison and Max Chernitsov for providing for the images reuse through a Creative Commons license.

References

BERNARDINI, F., MARTIN, I. M., AND RUSHMEIER, H. 2001. High-quality texture reconstruction from multiple scans. *IEEE Trans. on Visualization and Computer Graphics 7*, 4, 318–332.

BREGLER, C., MALIK, J., AND PULLEN, K. 2004. Twist based acquisition and tracking of animal and human kinematics. *Int. J. Comput. Vision 56*, 3, 179–194.

BROX, T., ROSENHAHN, B., AND WEICKERT. 2007. Three-dimensional shape knowledge for joint image segmentation and pose tracking. *Int. J. Comput. Vision 73*, 3, 243–262.

COLOMBO, C., BIMBO, A. D., AND PERNICI, F. 2005. Metric 3D reconstruction and texture acquisition of surfaces of revolution from a single uncalibrated view. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 1, 99–114.

DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. 1996. Modeling and rendering architecture from photographs:. Tech. rep., Berkeley, CA, USA.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum 21*, 3, 209–218.

GINGOLD, Y. I., DAVIDSON, P. L., HAN, J. Y., AND ZORIN, D. 2006. A direct texture placement and editing interface. In *UIST*, 23–32.

HARTLEY, R. I., AND ZISSERMAN, A. 2004. *Multiple View Geometry in Computer Vision*. Cambridge University Press.

HORMANN, K., LÉVY, B., AND SHEFFER, A. 2007. Mesh parameterization: theory and practice. In *SIGGRAPH courses*.

KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: constructing constrained texture maps. In *SIGGRAPH*, 326–333.

LEE, T., YEN, S., AND YEH, I. 2008. Texture mapping with hard constraints using warping scheme. *IEEE Trans. on Visualization and Computer Graphics 14*, 2, 382–395.

LENSCH, H. P. A., HEIDRICH, W., AND SEIDEL, H. 2000. Automated texture registration and stitching for real world models. In *Pacific Graphics*, 317.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH*, 362–371.

(a) Model (8k faces) (b) Source images and mapping (c) Result

Figure 13: *Texturing an entire model of a woman using 75 user-defined features.*

Figure 14: *Texturing a man using a variety of casual images, in which the men exhibit different proportions and articulations.*

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *SIGGRAPH*, 417–424.

MURRAY, R. M., SASTRY, S. S., AND ZEXIANG, L. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA.

SCHAEFER, S., MCPHAIL, T., AND WARREN, J. 2006. Image deformation using moving least squares. *ACM Trans. on Graphics 25*, 3, 533–540.

SCHMIDT, R., GRIMM, C., AND WYVILL, B. 2006. Interactive decal compositing with discrete exponential maps. *ACM Trans. on Graphics 25*, 3, 605–613.

SINHA, S., STEEDLY, D., SZELISKI, R., AGRAWALA, M., AND POLLEFEYS, M. 2008. Interactive 3D architectural modeling from unordered photo collections. *ACM Trans. on Graphics 25*, 5, 159:1–10.

TAI, Y.-W., BROWN, M., TANG, C.-K., AND SHUM, H.-Y. 2008. Texture amendment: reducing texture distortion in constrained parameterization. *ACM Trans. on Graphics 27*, 5, 1–6.

WEINHAUS, F. M., AND D., V. 1997. Texture mapping 3D models of real-world scenes. *ACM Comput. Surv. 29*, 4, 325–365.

WEINHAUS, F., AND DEVICH, R. 1999. Photogrammetric texture mapping onto planar polygons. *Graphical Models and Image Processing 61*, 2, 63–83.

XIAO, J., FANG, T., TAN, P., ZHAO, P., OFEK, E., AND QUAN, L. 2008. Image-based facade modeling. *ACM Trans. on Graphics 25*, 5, 161:1–10.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2005. Feature-based surface parameterization and texture mapping. *ACM Trans. on Graphics 24*, 1, 1–27.

ZHOU, K., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM, H. 2005. TextureMontage: seamless texturing of arbitrary surfaces from multiple images. *ACM Trans. on Graphics 24*, 3, 1148–1155.