Constrained Texture Mapping for Polygonal Meshes

Bruno Lévy*

ISA (Inria Lorraine and CNRS), France

Abstract

Recently, time and effort have been devoted to automatic texture mapping. It is possible to study the parameterization function and to describe the texture mapping process in terms of a functional optimization problem. Several methods of this type have been proposed to minimize deformations. However, these existing methods suffer from several limitations. For instance, it is difficult to put details of the texture in correspondence with features of the model, since most of the existing methods can only constrain iso-parametric curves.

We introduce in this paper a new optimization-based method for parameterizing polygonal meshes with minimum deformations, while enabling the user to interactively define and edit a set of constraints. Each user-defined constraint consists of a relation linking a 3D point picked on the surface and a 2D point of the texture. Moreover, the non-deformation criterion introduced here can act as an extrapolator, thus making it unnecessary to constrain the border of the surface, in contrast with classic methods. To minimize the criterion,a conjugate gradient algorithm is combined with a compressed representation of sparse matrices, making it possible to achieve a fast convergence.

CR Categories: I.3.3 [Computer Graphics] Picture/Image Generation ; I.3.5 [Computer Graphics]: Three-Dimentional Graphics and Realism—Color, shading, shadowing and texture; I.4.3 [Image processing]: Enhancement—Geometric Correction, Texture

Keywords: Texture Mapping, Paint Systems, Polygonal Modeling

*levy@loria.fr

ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA

1 INTRODUCTION

Texture mapping is a technique commonly used in computer graphics to generate realistic and visually rich images, by putting each object in correspondence with a 2D image. The notion of parameterization provides a mathematical formalism for studying this problem [16, 15]. A parameterization is a function putting the 3D surface to be textured in one-to-one correspondence with a subset of $\mathbb{R}^2$, called the *parameter space*. When such a parameterization is associated with a surface, it is possible to texture-map the surface by painting its parameter space with an image. For instance, Catmull has proposed to apply this technique to bi-cubic splines in [4]. This latter approach does not provide any means of controlling the nature and the repartition of the deformations. For instance, in the case of a sphere provided with its natural parameterization, high deformations are likely to occur near the poles.

1.1 Previous Work

Different approaches have been described to improve these results, such as using simple geometric transforms [3, 22], unfolding the surfaces [2, 9, 24] or global optimization approaches [14, 17]. Maillot *et. al.* propose in [18] to minimize a norm of the Green-Lagrange deformation tensor, but the repartition of the deformations are still difficult to control when using this method. Pedersen introduces in [23] a global optimization method, but the times reported seem to indicate that a great deal of user interaction is required, as well as in [14], where the user needs to construct a large number of iso-parametric curves. In [11], the notion of *conformal map*, from differential geometry, is used to generate an angle-preserving parameterization.

The theory of planar graphs provides a mathematical background for studying the parameterization problem. For instance, an approximation of harmonic maps [6] has been used by Eck *et. al.* in [5] to parameterize triangulated surfaces previously decomposed into topological disks (the same kind of approach based on subdivision is described in [15]). Similarly, the notion of barycentric application [25] has been used in [7] for the same purpose. The problem of minimizing the deformations is addressed there by defining appropriate weights for the convex combinations involved in the barycen-

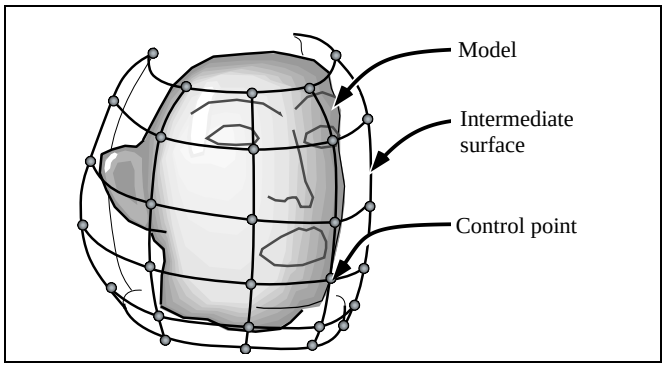


Figure 1: An indirect and partial solution to the problem of feature matching consists in surrounding the model with an intermediate parametric surface, painted with the texture. The user can then interactively edit the intermediate surface, by moving its control points. Texture coordinates on the model are retrieved by projecting them from the intermediate surface.

tric map. The approach introduced in [16] combines the notion of barycentric application with penalty functions to preserve right angles and to ensure an homogeneous spacing of the iso-parametric curves all over the surface. In [12, 13], a non-deformation criterion is introduced, with extrapolating capabilities. In other words, using this criterion, it is not necessary to constrain the border of the surface. However, minimizing this latter criterion requires a non-linear solver, which may result in expensive computations.

1.2 Matching features

The different approaches mentioned above make it possible to map an image onto an object, with more or less user interaction required. Using these approaches, it is possible to construct a parameterization presenting satisfying non-deformation properties in a reasonable time. However, the problem of matching features between the model and the texture has not been addressed by these methods. In the specific case where the texture and the model are acquired from the same source (e.g. a 3D scanner), the problem of feature matching becomes a calibration problem. More generally, we consider here the case where the texture and the model come from a different source. In this case, the texture needs to be warped on the model according to user-specified information. For instance, when considering a face, the user may want to specify a correspondence between the eyes on the model and the eyes on the texture. A solution for matching iso-parametric curves is proposed in [16]. Unfortunately, this method cannot be applied in the general case, since it is not always possible to constrain the features with iso-parametric curves. For instance, it is impossible to map a closed curve (such as the border of an eye) to an iso-parametric curve.

As shown in Figure 1, the problem of feature matching is addressed in the industry in an **indirect** way. Using this approach, when a specific feature needs to be matched, it is very likely that the user needs to edit several control points of the intermediate surface. Therefore, a large number of trials and errors is required to realize a texture mapping which precisely matches the features of the model. A better approach to feature matching consists of parameterizing the model without any constraint, and then warping the texture in 2d, as done in [10]. In the approach we present here, we consider that 3d-2d feature matching is as simple as 2d-2d morphing, and therefore can be performed **directly**, during the parameterization process.

1.3 Overview

The approach presented in this paper enables a **direct** manipulation of the mapping, and defines a regularization criterion ensuring a smooth transition between the constraints specified by the user. As shown further, such a constraint is a link between a 3d point of the model and a pixel in texture space, and both can be interactively modified.

Compared to other methods, our approach presents the following advantages:

- an arbitrary set of constrained features can be honored in the least squares sense, whereas other methods can only constrain iso-parametric curves (e.g. [15, 16]) ;
- the non-deformation criterion we introduce can act as an extrapolator, which means that it is not necessary to constrain the border. Therefore, more degrees of freedom are available to the system for minimizing the deformations. In contrast with the method proposed in [12], the criterion can be minimized by a linear solver ;
- the solver introduced here is based on the conjugate gradient method and a sparse representation of matrices. It has a better rate of convergence than methods based on successive over relaxations, such as in [16]. It can incorporate the constraints presented there, such as those enabling to take discontinuities into account.
- using our approach, it is unnecessary to triangulate the polygons of the model, since any open surface composed of convex polygons can be used (non-convex polygons can be preprocessed by the classical *ear-cutting* algorithm).

The paper is organized as follows. The next section introduces the notion of parameterization for a polygonal mesh. In Section 3, it is shown how to express the constrained parameterization problem in terms of quadratic optimization. Section 4 details the numerical solver and gives an analysis of its space and time complexities. A few examples are shown at the end of the paper.

2 PARAMETERIZATION OF POLYGONAL MESHES

We consider here the case of a surface homeomorphic to a disc. More complex surfaces can be decomposed into a set of tiles, using texture-atlas approaches, as done in [21],[15] and [5]. In this case, our approach may be used to parameterize each tile of the atlas.

2.1 Definitions

As shown in Figure 2, given an open surface $\mathbf{S}$ of $I\!R^3$, a mapping $\mathcal{U}$ is a one-to-one transform, putting the surface $\mathbf{S}$ in correspondence with a subset Ω of $I\!R^2$.

$$(x,y,z) \in \mathbf{S} \quad \rightarrow \quad \mathcal{U}(x,y,z) = \begin{bmatrix} \mathcal{U}_u(x,y,z) \\ \\ \mathcal{U}_v(x,y,z) \end{bmatrix}$$

In the remainder of this paper, the following terms will be used:

- The set Ω is called the (u,v) *parameter space*;
- The inverse function $\mathcal{X} = \mathcal{U}^{-1}$ is called a *parameterization* of the surface:

$$(u,v) \in \Omega \quad \rightarrow \quad \mathcal{X}(u,v) = \mathcal{U}^{-1}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}$$

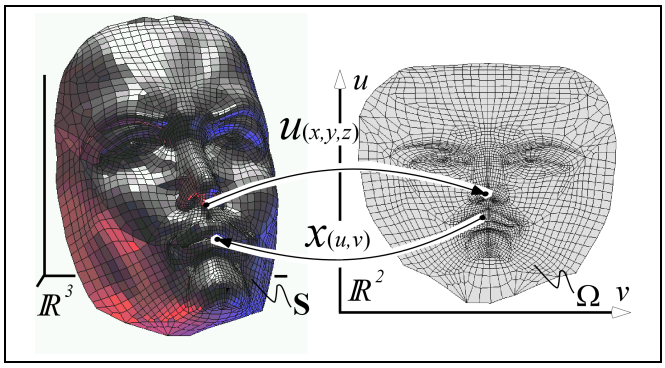


Figure 2: A mapping $\mathcal{U}()$ puts a surface $\mathbf{S}$ of $\mathbb{R}^3$ in one-to-one correspondence with a subset Ω of $\mathbb{R}^2$ called the (u, v) parameter space. The inverse $\mathcal{X}()$ of $\mathcal{U}()$ is named a parameterization of $\mathbf{S}$.

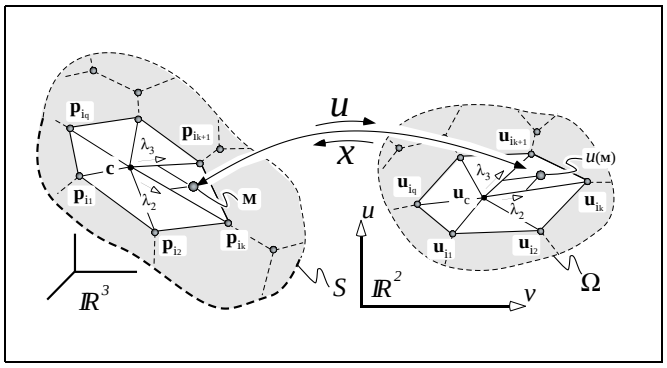


Figure 3: For a polygonal mesh, a mapping $\mathcal{U}()$ can be defined to be a piecewise linear function, defined by its values $\mathbf{u}_i = (u_i, v_i)$ at the vertices of the mesh. Each polygon is virtually decomposed into triangles over which the mapping is linearly interpolated.

2.2 Piecewise Linear Parameterization

In the case considered here, the surface $\mathbf{S}$ is represented by a polygonal mesh $\mathcal{G} = \{[1 \dots n], \{\mathcal{P}\}\}$, where $[1 \dots n]$ denotes the set of vertices identified by integers, and $\{\mathcal{P}\}$ the set of polygons. Each polygon $\mathcal{P} = \{i_1, i_2, \dots i_q\}$ is determined by the list of its vertices indices. The geometric location at the vertex i is denoted $\mathbf{p}_i$. For a surface represented by this type of discretization, it is possible to represent a parameterization by providing each vertex i with its image $\mathbf{u}_i = (u_i, v_i)$ through the mapping. Then, as shown in Figure 3, supposing that all the values (u_i, v_i) are known, the mapping can be defined as a piecewise linear function, by virtually decomposing each polygon $\mathcal{P}$ into a set of triangles incident to the center $\mathbf{c}$ of $\mathcal{P}$. Given a point $\mathbf{M}$ in the polygon $\mathcal{P}$, the mapping $\mathcal{U}(\mathbf{M})$ at $\mathbf{M}$ is given by :

$$\mathcal{U}(\mathbf{M}) = \lambda_1.\mathbf{u_c} + \lambda_2.\mathbf{u}_{i_k} + \lambda_3.\mathbf{u}_{i_{k+1}} \tag{1}$$

where :

- $\mathbf{c} = 1/q.\sum_l \mathbf{p}_{i_l}$ is the center of $\mathcal{P}$ and $\mathbf{u_c} = 1/q.\sum_l \mathbf{u}_{i_l}$, its image in parameter space, where q denotes the number of vertices of $\mathcal{P}$;
- k and $k+1$ are such that $\{\mathbf{c}, \mathbf{p}_{i_k}, \mathbf{p}_{i_{k+1}}\}$ is the unique (virtual) sub-triangle of $\mathcal{P}$ that contains $\mathbf{M}$;
- $(\lambda_1, \lambda_2, \lambda_3)$ are the barycentric coordinates at $\mathbf{M}$ in $\{\mathbf{c}, \mathbf{p}_{i_k}, \mathbf{p}_{i_{k+1}}\}$.

Conversely, the piecewise linear parameterization $\mathcal{X}$ at a point $\mathbf{u} = (\mathbf{u}_u, \mathbf{u}_v)$ in a triangle $\{\mathbf{u_c}, \mathbf{u}_{i_k}, \mathbf{u}_{i_{k+1}}\}$ of the parameter space Ω is given by :

$$\mathcal{X}(\mathbf{u}) = \lambda_1.\mathbf{c} + \lambda_2.\mathbf{p}_{i_k} + \lambda_3.\mathbf{p}_{i_{k+1}} \tag{2}$$

where λ and μ are the barycentric coordinates at $\mathbf{u}$ in $\{\mathbf{u}_{i_k}, \mathbf{u}_{i_{k+1}}, \mathbf{u_c}\}$. It is easy to check that in a given triangle, the so-defined $\mathcal{X}$ function corresponds to the inverse of the mapping $\mathcal{U}$.

3 CONSTRAINED TEXTURE MAPPING

Given this representation of the mapping function, stored in a polygonal mesh, our purpose is now to provide texture mapping with the equivalent of data fitting approaches existing in the realm of geometric design. In this field, surfaces are often represented by (polynomial) functions. When a surface represented by a function $\mathcal{X}$ is supposed to pass through a set of m data points $\mathbf{M}_j$ associated with the parameter-space points $\mathbf{U}_j$, objective functions like the following $C(\mathcal{X})$ are often minimized :

$$C(\mathcal{X}) = \sum_{j=1}^{m} \{\mathbf{M}_j - \mathcal{X}(\mathbf{U}_j)\}^2 + \epsilon \int_\Omega \left(\frac{\partial^2 \mathcal{X}}{\partial u^2}\right)^2 + \left(\frac{\partial^2 \mathcal{X}}{\partial v^2}\right)^2 du.dv \tag{3}$$

In this equation, the first term represents the squared deviation at the data points, and the second term enforces the smoothness of the solution. The user-defined parameter $\epsilon \in]0, \infty[$ makes it possible to choose a compromise between the accuracy of the fitting and the smoothness of the solution. In our case, it is easier to characterize the mapping function rather than the parameterization. The next section shows how to define a similar criterion for such a function, defined over a polygonal mesh, to enable a direct manipulation of the mapping (see Figure 4). It will be also shown how to constrain the gradients of the mapping. Then, how to minimize the so-defined criterion will be explained.

3.1 Matching Features

Each feature point is defined to be a couple of points $(\mathbf{M}_j, \mathbf{U}_j)$, where $\mathbf{M}_j \in \mathbb{R}^3$ is a point belonging to the surface $\mathbf{S}$, and $\mathbf{U}_j = (U_j, V_j)$ the desired texture coordinates at $\mathbf{M}_j$. The data fitting term of Equation 3 becomes :

$$C_{\text{fit}} = \sum_{j=1}^{m} \Delta^2_{\text{fit}}(\mathbf{M}_j) = \sum_{j=1}^{m} \|\mathbf{U}_j - \mathcal{U}(\mathbf{M}_j)\|^2$$

Recalled from Equation 1, the mapping $\mathcal{U}$ at $\mathbf{M}$ is a linear combination of the values $\mathbf{u}_{i_l}$ at the vertices of the polygon $\mathcal{P} = \{i_1, \dots i_q\}$ that contains $\mathbf{M}$. Then, by replacing the center of the polygon in parameter space $\mathbf{u_c}$ by its expression, the fitting term $\Delta^2_{\text{fit}}(\mathbf{M}_j)$ is defined by the coefficients a_i :

$$\Delta^2_{\text{fit}}(\mathbf{M}_j) = \left(U_j - \sum_{i=1}^{n} a_i.u_i\right)^2 + \left(V_j - \sum_{i=1}^{n} a_i.v_i\right)^2$$

$$\text{where :} \begin{cases} a_{i_k} = \lambda_1/q + \lambda_2 & \\ a_{i_{k+1}} = \lambda_1/q + \lambda_3 & \\ a_i = \lambda_1/q & \forall i \in \mathcal{P} - \{i_k, i_{k+1}\} \\ a_i = 0 & \text{everywhere else} \end{cases} \tag{4}$$

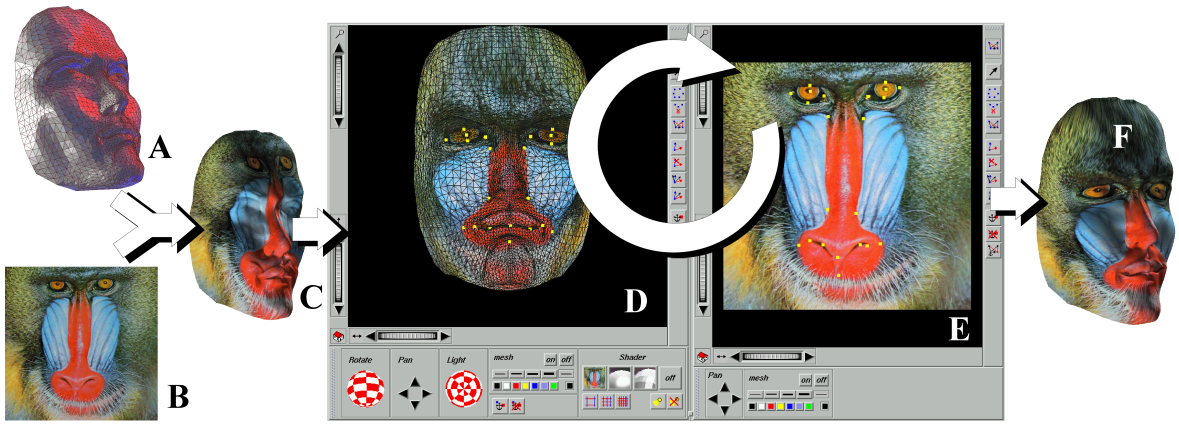


Figure 4: Outline of the method : Starting from a polygonal mesh (A) and an image (B), an initial parameterization is automatically computed (C). The user can iteratively refine it, by creating constraints, and editing them either in model space (D), or in texture space (E). The object can be updated and displayed after each modification of the constraints (update time: 2 s.) ; F : Final result (total session time : 5 min.)

In this equation, as in the previous section, i_k and i_{k+1} denote the vertices defining the virtual sub-triangle of $\mathcal{P}$ that contains $\mathbf{M}_j$ (see Figure 3). This generalizes Mallet's *fuzzy control points* [19].

3.2 Constraining the Gradient

Controlling the gradient of the parameterization provides an additional way to interact with the texture mapping process. As shown in Figure 5, this provides editing capabilities similar to "free-form design". In addition, the involved equations will be also used further to define the regularization term.

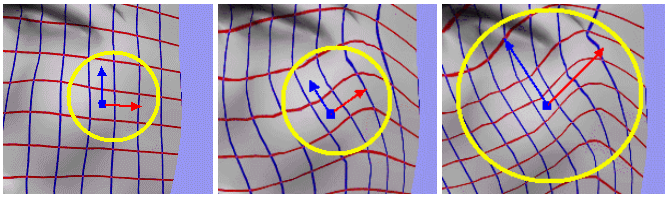


Figure 5: Editing the gradients of a parameterization. A: Initial configuration; B: Editing the direction of the gradients; C: Editing their magnitude.

Given a triangle $\mathcal{T} = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ in $I\!R^3$, and given three values ν_1, ν_2, ν_3 associated with the vertices of $\mathcal{T}$, the gradient of the linear interpolation of ν over $\mathcal{T}$ is constant, and will be denoted $\text{grad}(\nu|\mathcal{T})$. Its components are linear combinations of the values (ν_1, ν_2, ν_3). The gradient will be expressed in the local orthonormal basis $(\mathbf{p}_1, \mathbf{X}, \mathbf{Y})$:

$$\mathbf{X} = \frac{\mathbf{p}_2 - \mathbf{p}_1}{\|\mathbf{p}_2 - \mathbf{p}_1\|} \quad ; \quad \mathbf{Y} = \frac{\mathbf{X} \times (\mathbf{p_3} - \mathbf{p_1}) \times \mathbf{X}}{\|\mathbf{X} \times (\mathbf{p_3} - \mathbf{p_1}) \times \mathbf{X}\|}$$

In this basis, the gradient is given as follows (see [20]) :

$$\text{grad}(\nu|\mathcal{T}) = \left[\sum_{i=1}^{3} TX_i.\nu_i \quad \sum_{i=1}^{3} TY_i.\nu_i \right]^t$$

$$\left\{ \begin{array}{llllll} TX_1 & = & (Y_2 - Y_3)/d & ; & TY_1 & = & (X_2 - X_3)/d \\ TX_2 & = & (Y_3 - Y_1)/d & ; & TY_2 & = & (X_3 - X_1)/d \\ TX_3 & = & (Y_1 - Y_2)/d & ; & TY_3 & = & (X_1 - X_2)/d \\ \multicolumn{6}{l}{d = (X_2 - X_1).(Y_3 - Y_1) - (X_3 - X_1).(Y_2 - Y_1)} \end{array} \right. \quad (5)$$

In this equation, (X_i, Y_i) denote the coordinates at $\mathbf{p}_i$ in the basis $(\mathbf{p}_1, \mathbf{X}, \mathbf{Y})$. Given a point $\mathbf{M}_j$ and a vector $\mathbf{U}_j$ specified by the user to constrain the gradients of u, and $\mathcal{P} = \{i_1, i_2, \ldots i_q\}$, the polygon that contains $\mathbf{M}_j$, the squared deviation $\Delta^2_{\text{grad}}(\mathbf{M}_j)$ of the gradient is given by :

$$\begin{aligned} \Delta^2_{\text{grad}}(\mathbf{M}_j) &= \|\mathbf{U}_j - \text{grad}_u(\mathbf{M}_j)\|^2 \\ &= \left(\mathbf{U}_j.\mathbf{X} - \sum_{i=1}^{n} a_i.u_i\right)^2 + \left(\mathbf{U}_j.\mathbf{Y} - \sum_{i=1}^{n} a'_i.u_i\right)^2 \end{aligned}$$

where :

$$\left\{ \begin{array}{llllll} a_{i_k} & = & TX_1/q + TX_2 & ; & a'_{i_k} & = & TY_1/q + TY_2 \\ a_{i_{k+1}} & = & TX_1/q + TX_3 & ; & a'_{i_{k+1}} & = & TY_1/q + TY_3 \\ \multicolumn{6}{l}{\forall i \in \mathcal{P} - \{i_k, i_{k+1}\}, \quad a_i = TX_1/q \quad ; \quad a'_i = TY_1/q} \\ \multicolumn{6}{l}{\forall i \notin \mathcal{P}, \quad a_i = 0 \quad ; \quad a'_i = 0} \end{array} \right. \quad (6)$$

The same term can be defined to constrain the gradient of v.

3.3 Regularization

In Equation 3, expressing both the data fitting and regularization criteria, the parameterization $\mathcal{X}$ is characterized. In our case, since the unknowns are the (u_i, v_i) texture coordinates, it is easier to characterize the mapping function $\mathcal{U}$. Our goal is then to adapt the regularization criterion to a piecewise linear mapping function $\mathcal{U}$. The regularization criterion, as defined in Equation 3, involves the second derivatives of the parameterization. Since the second derivatives are not defined for a piecewise linear parameterization, we need to define an equivalent criterion. The second order derivatives may be thought of as the variation of the gradient.

As shown in Figure 6, a possible regularization criterion can then be defined as the sum of the gradient variations over all edges $\mathcal{E}$ of the mesh. Considering two adjacent polygons $\mathcal{P} = \{i_1, \ldots i_q\}$ and $\mathcal{P}' = \{i'_1, \ldots i'_{q'}\}$, the variation of the gradient between $\mathcal{T} = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{c}\}$ and $\mathcal{T}' = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{c}'\}$ is given by :

$$C_{\text{reg}} = \sum_{i_1, i_2 \in \mathcal{E}} \Delta^2_{\text{reg}}(i_1, i_2)$$

where :

$$\begin{aligned} \Delta^2_{\text{reg}}(i_1, i_2) &= \{\text{grad}(u|\mathcal{T}) - \text{grad}(u|\mathcal{T}')\}^2 \\ &+ \{\text{grad}(v|\mathcal{T}) - \text{grad}(v|\mathcal{T}')\}^2 \end{aligned}$$

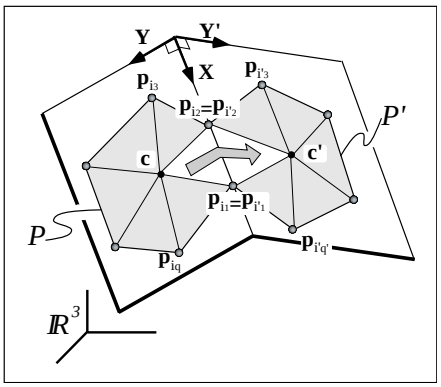
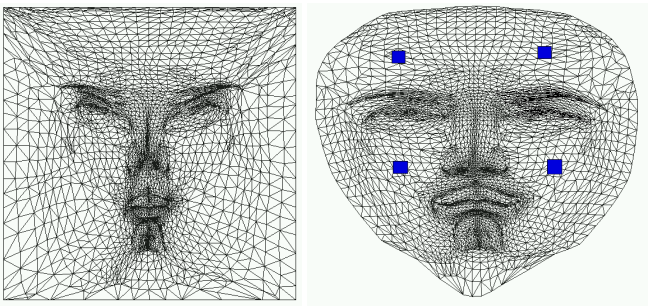


Figure 6: Minimizing the variations of the gradient between two adjacent polygons.



A B

Figure 7: A : When using harmonic maps, the border needs to be fixed, which can cause high deformations ; B : Our criterion can extrapolate a parameterization from any set of fixed points (shown as squares), and does not need the border to be fixed.

In this equation, the gradient can be computed using its expression, given by Equation 5. As can be seen, the term TX_3 is null, which means that the X component of the gradient only depends on ν_1 and ν_2, and is therefore the same for two triangles sharing the edge $\mathbf{p}_1, \mathbf{p}_2$. For this reason, only the Y component plays a role in the regularization criterion. The variation of the gradient can then be expressed as follows :

$$\Delta^2_{\text{reg}}(i_1, i_2) = \left\{ \sum_{i=1}^{n} a_i.u_i \right\}^2 + \left\{ \sum_{i=1}^{n} a_i.v_i \right\}^2$$

where :

$$\left\{ \begin{array}{lcl} a_{i_1} & = & TY_1/q + TY'_1/q' + TY_2 + TY'_2 \\ a_{i_2} & = & TY_1/q + TY'_1/q' + TY_3 + TY'_2 \\ \forall i \in \mathcal{P} - \{i_1, i_2\}, & a_i & = TY_1/q \\ \forall i \in \mathcal{P}' - \{i_1, i_2\}, & a_i & = TY'_1/q' \\ \forall i/i \notin \mathcal{P}, i \notin \mathcal{P}', & a_i & = 0 \end{array} \right. \quad (7)$$

Figure 7-A shows an harmonic map. Such a parameterization requires the border of the surface to be fixed, which can cause deformations. Note that a better boarder curve can be chosen, to obtain a less deformed result, but this requires in practice a great deal of user interaction, especially when the border has a complex shape. In contrast, our regularization criterion can act as an extrapolator. As shown in Figure 7-B, a parameterization can be extrapolated from an arbitrary set of fixed points (in blue).

The next section shows how to minimize the global criterion, combining the regularization and the fitting terms.

4 NUMERICAL OPTIMIZATION

As can be seen by summing the terms from Equations 4, 6 and 7, defining respectively the point fitting, gradient fitting, and regularization terms, the criterion $C(\mathcal{U})$ is a sum of squared linear relations, and can be therefore written as follows :

$$\begin{aligned} C(\mathcal{U}) &= \sum_{j=1}^{m} \Delta^2_{\text{fit}}(\mathbf{M}_j) + \sum_{j=1}^{m'} \Delta^2_{\text{grad}}(\mathbf{M}_j) + \epsilon \sum_{e \in \mathcal{E}} \Delta^2_{\text{reg}}(e) \\ &= \sum_k \left(b_k - \sum_{i=1}^{2.n} a_{k,i}.x_i \right)^2 = \|A.\mathbf{x} - \mathbf{b}\|^2 \end{aligned}$$

In this equation, the vector $\mathbf{x} \in \mathbb{R}^{2.n}$ is the vector of all unknowns (u_i, v_i), defined by $x_i = u_i$ and $x_{i+n} = v_i$. The k^{th} squared linear relation of the criterion C yields a row $\{a_{k,1}, \ldots a_{k,2.n}\}$ of the matrix A, and a component b_k of the vector $\mathbf{b}$. Each row $\{a_{k,i}\}, b_k$ is normalized, and ϵ is set to 10 in the examples shown here. Solving the following problem is equivalent to minimize the criterion C :

$$\left| \begin{array}{l} \text{minimize } \frac{1}{2}.\mathbf{x}^t.G.\mathbf{x} + \mathbf{c}^t.\mathbf{x} \\ \\ \text{where } G = A^t.A \quad ; \quad \mathbf{c} = -A^t.\mathbf{b} \end{array} \right.$$

It is possible to ensure the existence and uniqueness of the minimum by adding a *roughness* term to this equation, as shown in [19].

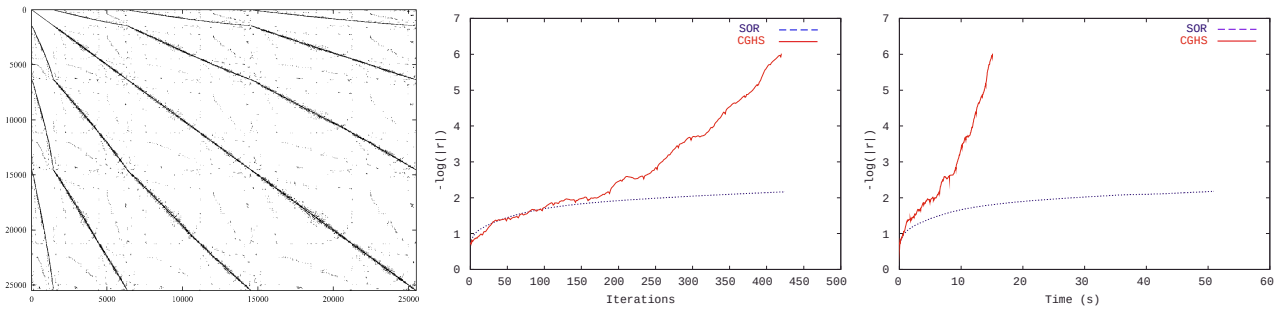
4.1 Conjugate Gradient for Sparse Matrices

This minimization problem can be easily solved by the conjugate gradient method (see e.g. [8]). The variant by Hestenes and Stiefel of this method [1] is very efficient, and can be implemented as follows. Starting from an initial solution $\mathbf{x}$, the following algorithm iteratively minimizes the norm of the residual $\|G.\mathbf{x} + \mathbf{c}\|$ and stops when it is smaller than $\varepsilon\|\mathbf{c}\|$:

solve(G, $\mathbf{x}$, $\mathbf{c}$, ε) :
Variables :
$\mathbf{g}, \mathbf{r}, \mathbf{p} \in \mathbb{R}^{2.n}$
$t, \tau, \sigma, \rho, \gamma$, treshold $\in \mathbb{R}$

treshold $\leftarrow \varepsilon^2.\|\mathbf{b}\|^2$; $\mathbf{g} \leftarrow -(G.\mathbf{x} + \mathbf{c})$; $\mathbf{r} \leftarrow \mathbf{g}$
while $\|\mathbf{g}\|^2 >$ treshold
$\mathbf{p} \leftarrow G.\mathbf{r}$
$\rho \leftarrow \|\mathbf{p}\|^2$; $\sigma \leftarrow \mathbf{r}.\mathbf{p}$; $\tau \leftarrow \mathbf{g}.\mathbf{r}$; $t \leftarrow \tau/\sigma$
$\mathbf{x} \leftarrow \mathbf{x} + t.\mathbf{r}$; $\mathbf{g} \leftarrow \mathbf{g} - t.\mathbf{p}$
$\gamma \leftarrow (t^2.\rho - \tau)/\tau$; $\mathbf{r} \leftarrow \gamma.\mathbf{r} + \mathbf{g}$
end // *while*

It can be noticed that the main loop applies simple vector operations, and a single matrix-vector product per iteration. The matrix $G = A^t.A$ is sparse, and can be stored in compressed form : each row i of G is then represented by a list of couples $\{(j, g_{i,j})\}$, where only non-zero entries are represented. This representation of G allows an efficient implementation of the matrix-vector product involved in the main loop of the algorithm. Knowing the set of relations $\{a_{k,i}\}, b_k$, it is easy to iteratively construct the matrix $G = A^t.A$ in compressed form, and the vector $\mathbf{c} = -A^t.\mathbf{b}$, by adding the contribution of each term to the corresponding coefficient.



A B C

Figure 8: A : non-zero entries of the $G = A^t A$ matrix. Each row has $(2\bar{q} - 2)^2$ non-zero coefficients, where $\bar{q}$ denotes the average number of vertices per polygon ; B,C : Compared convergence of our method (continuous red) and a relaxation based method (dashed blue), in function of iterations and time, respectively.

4.2 Time and Space Complexity

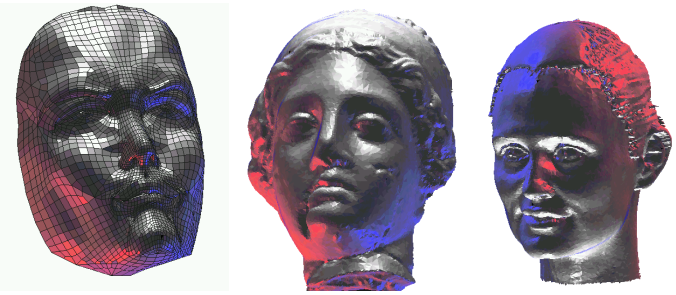
Since each edge of the mesh yields a regularization term, the coefficients yielded by the C_{fit} and C_{grad} terms can be neglected in the complexity analysis. If we consider a mesh with an average of $\bar{q}$ vertices per polygon, then each row of A corresponding to a $C_{\text{reg}(i_1,i_2)}$ term has $2.\bar{q} - 2$ non-zero coefficients (the number of vertices shown in figure 6). Then, the matrix $G = A^t.A$ has $(2.\bar{q} - 2)^2$ non-zero coefficient per row. Theoretically, for this system of dimension $2.n$, the conjugate gradient algorithm converges in $2.n$ iterations at most. From this simple analysis, it can be concluded that :

- Theoretically, the optimum is found in $O(\bar{q}^2.n^2)$ operations (as shown further, it can be much less in practice) ;
- The matrix G and the vector $\mathbf{c}$ are constructed in $O(\bar{q}^2.n)$ operations ;
- The matrix G requires $O(\bar{q}^2.n)$ words of memory.

In Figure 8, the results of numerical experiments are shown, using a triangulated surface having 13000 vertices. Figure 8-A shows the non-zero coefficients of the $A^t.A$ matrix. In Figures 8-B and C, the convergence of our method is studied, and compared with the successive over relaxation solver (SOR) used in [16]. The curves show the log of the residual norm, in function of the iterations and time respectively. As can be seen, our method converges in less iterations. The method used in [16] consists of a SOR method, where the coefficients of G are evaluated on the fly. In contrast, in our method, since the matrix G is pre-computed, each iteration takes less time. Various experiments have shown that a tolerance $\varepsilon = 10^{-5}$ for the norm of the residual gives visually accurate results. With our method, for the 13000 vertices triangulated surface, this threshold is reached after 374 iterations, in 12 seconds.

Statistics for various data sets are shown in Figure 9 (*face hr* is a refined version of *face*). The initial parameterization is created by starting from a projection used as the initial vector $\mathbf{x}_0$ for the algorithm. As can be seen in the table, a non-distorting parameterization is created in much less iterations than the theoretical number (i.e. $2.n$). Using this method, a mesh made of 50k triangles can be parameterized in less than 100 seconds on a 400 Mhz Pentium II. The table shows also the average time spent in updating the solution when the set of constraints is modified.

As far as memory is concerned, a surface of 50k triangles requires 6.5 megabytes of memory for the compressed representation of the system. For such a mesh, this representation is constructed in less than 2 seconds. For larger surfaces, if the system does not fit in memory, it is possible to consider the mesh as a compressed representation of the matrix A, and compute the coefficients of G on the fly. Then, solving the system is supposed to take $(2.\bar{q} - 2)$ times more time than when G is stored, which has been confirmed by our experiments.



A B C

dataset	♯ vrtx.	♯ facets	$\bar{q}$	G (s)	♯ iter. initial	initial (s)	update (s)
A: face	3254	3209	$\simeq 4$	0.15	558	7.62	2
face hr	12917	12818	$\simeq 4$	0.83	706	43	8
B: Higea	18842	37279	3	1.03	688	65.8	12
C: woman	25298	50063	3	1.34	735	95.7	16

Figure 9: Statistics of the solver, applied to various data sets. For each data set, the table shows the time to construct the matrix G, the number of iterations and time to compute the initial parameterization, and the time to update the solution when a constraint is modified. These times have been measured using a 400 Mhz Pentium II. Textured models are shown further.

5 RESULTS

Several examples are shown in the figures below. Figure 10 shows our method applied to the classical *Bunny* data set. For this data set, the obtained mapping is valid (i.e. one-to-one), but deformations appear at the level of the ears. For this type of mesh, it is preferable to use our approach to parameterize the charts of a texture-atlas. It can be also noticed that our method does not guarantee the absence of overlaps. In our experiments, they were seldom encountered, and appeared in zones of high negative curvature, such as near the horns of the cow. In this case, it is easy to manually fix the problem, by adding a constraint. Despite these limitations, our method is easy to use, and has been successfully applied to various data sets. In Figure 11, the mouth is treated using the method presented in [16], to make the mapping continuous across it. For each example, the total session time is indicated (using a 400 Mhz Pentium II PC).

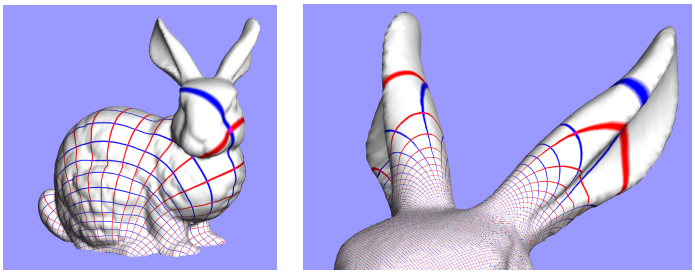


Figure 10: Parameterizing the *Bunny* data set.

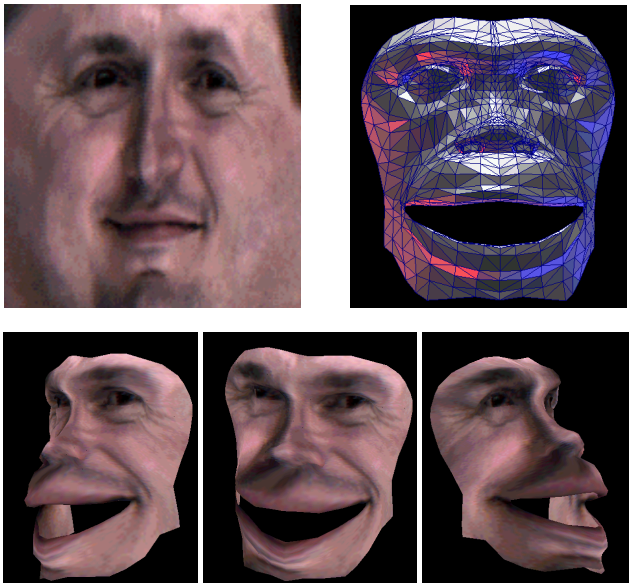


Figure 11: The *fran* image from the VTK toolkit mapped onto a monkey (total session time : 10 min.)

6 CONCLUSION

The methods presented in this paper allow new texturing tools to be implemented. Our approach may be thought of as a *morphing* tool, operating on curved tesselated surfaces. Various applications of the method can be imagined, in particular, we think that this method integrated in a 3D paint system (see e.g. [23]) could make the creation process easier, by freeing the artist from technicalities encountered in existing texturing packages. We will consider this latter aspect in future works. Applications to 3D gridding can also be imagined. For this type of applications, this tool could be incorporated into the Gocad software, a 3D modeler used in the oil and gas industry.

7 ACKNOWLEDGEMENTS

Thanks to Prof. Jean-Laurent Mallet, inventor of the D.S.I. method, who encouraged me to explore this type of approaches based on numerical optimization. Thanks also to his Gocad team. Thanks to Alias-Wavefront, Cyberware and Stanford for their 3D models. Thanks to the reviewer for their useful comments, and to the English reviewer for their corrections.

References

[1] Ashby, Manteuffel, and Saylor. A taxonomy for conjugate gradient methods. *J. Numer. Anal.*, 27:1542–1568, 1990.

[2] C. Bennis, J.M. Vézien, and G. Iglésias. Piecewise surface flattening for non-distorted texture mapping. In *SIGGRAPH Comp. Graph. Proc.*, volume 25, pages 237–246. ACM, July 1991.

[3] E. Bier and K. Sloan. Two-part texture mapping. *IEEE Computer Graphics and Applications*, pages 40–53, September 1986.

[4] E. Catmull. *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, Dept. of Computer Sciences, University of Utah, December 1974.

[5] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics (SIGGRAPH Conf. Proc.)*, pages 173–182. ACM, August 1995.

[6] J. Eells and J.H. Sampson. Harmonic mapping of riemannian manifolds. *Amer. J. Math.*, 86:109–160, 1964.

[7] M.S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, April 1997.

[8] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, 1981. ISBN 0-12-283950-1.

[9] J.P. Gratier, B. Guillier, and A. Delorme. Restoration and balance of a folded and faulted surface by best-fitting of finite elements: principles and applications. *Journal of Structural Geology*, 13(1):111–1115, 1991.

[10] Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Frédéric Pighin. Making faces. In Michael Cohen, editor, *Proceedings of SIGGRAPH 98*, Annual Conference Series, Addison Wesley, pages 55–66. Addison Wesley, 1998.

[11] S. Haker, S. Angenent, A. Tannenbaum, R. Kikinis, G. Sapiro, and M. Halle. Conformal surface parameterization for texture mapping. *IEEE Transactions on Visualization and Computer Graphics*, 6(2), April-June 2000.

[12] K. Hormann and G. Greiner. Mips: An efficient global parameterization method. In *Curve and surface design: saint-malo 1999*, pages 153–162. Vanderbilt university press, 2000.

[13] K. Hormann, G. Greiner, and S. Campagna. Hierarchical parameterisation of triangulated surfaces. In *Vision, Modeling and Visualization '99*, pages 219–226. infix, 1999.

[14] V. Krishnamurthy and M. Levoy. Fitting Smooth Surfaces to Dense Polygon Meshes. In *Computer Graphics (SIGGRAPH Conf. Proc.)*. ACM, August 1996.

[15] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. *Computer Graphics (Siggraph Conf. Proc)*, pages 95–104, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

[16] B. Levy and J.L. Mallet. Non-Distorted Texture Mapping for Sheared Triangulated Meshes. In *Computer Graphics (SIGGRAPH Conf. Proc.)*. ACM, July 1998.

[17] S.D. Ma and H. Lin. Optimal texture mapping. In *EUROGRAPHICS'88*, pages 421–428, September 1988.

[18] J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. In *SIGGRAPH Comp. Graph. Proc.*, volume 27. ACM, 1993.

[19] J.L. Mallet. Discrete smooth interpolation in geometric modeling. *ACM-Transactions on Graphics*, 8(2):121–144, 1989.

[20] J.L. Mallet. *Geomodeling*. Academic Press, to appear, 2001.

[21] Fabrice Neyret and Marie-Paule Cani. Pattern-based texturing revisited. In *SIGGRAPH 99 Conference Proceedings*. ACM SIGGRAPH, Addison Wesley, August 1999.

[22] Peachey and R. Darwyn. Solid texturing of complex surfaces. In *SIGGRAPH Comp. Graph. Proc.*, volume 19, pages 287–296. ACM, July 1985.

[23] H.K. Pedersen. Decorating implicit surfaces. In *SIGGRAPH Comp. Graph. Proc.*, pages 291–300. ACM, 1995.

[24] Samek, Marcel, C. Slean, and H. Weghorst. Texture mapping and distortions in digital graphics. *The Visual Computer*, 2(5):313–320, September 1986.

[25] W.T. Tutte. Convex representation of graphs. In *Proc. London Math. Soc.*, volume 10, 1960.

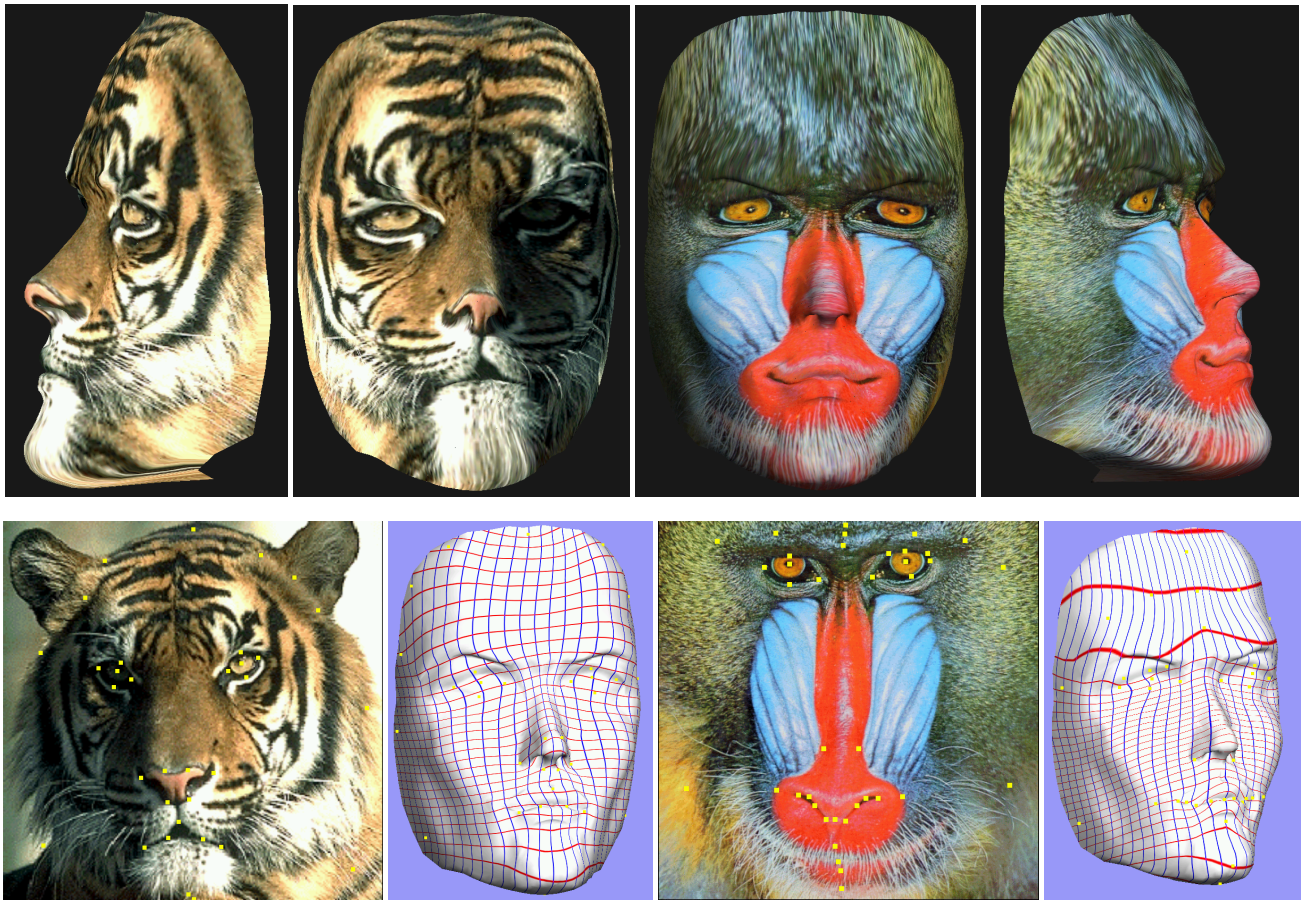


Figure 12: Mapping two different images onto the *Face* data set. The texture and iso-parametrics are shown for both. As can be seen, the mapping smoothly interpolates and extrapolates the feature points. (total session time : 5 min. for each)

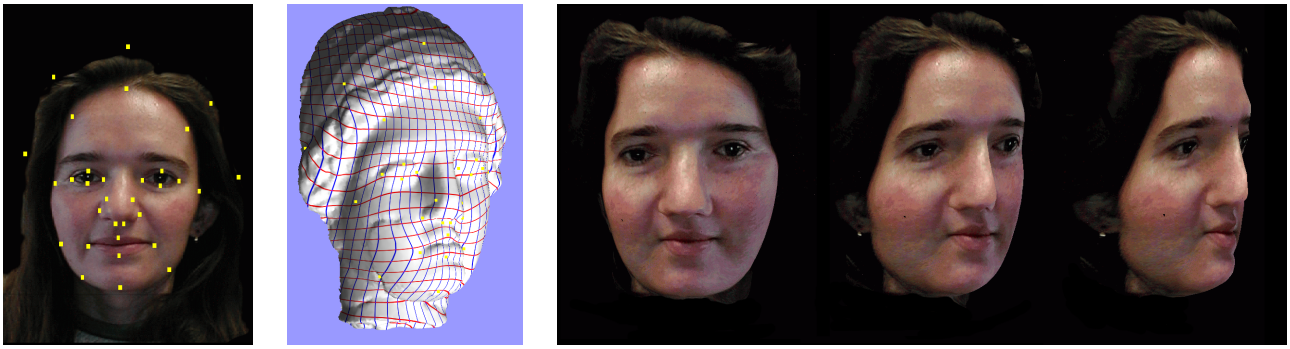


Figure 13: Texturing the *Higea* data set. For this data set, 32 feature points have been specified. (total session time : 15 min.)

Figure 14: A cow textured with a tiger and a young cheetah (total session time : 10 min. for each)