What Shape are Dolphins? Building 3D Morphable Models from 2D Images

Thomas J. Cashman and Andrew W. Fitzgibbon, *Senior Member, IEEE*

Abstract—3D morphable models are low-dimensional parametrizations of 3D object classes which provide a powerful means of associating 3D geometry to 2D images. However, morphable models are currently generated from 3D scans, so for general object classes such as animals they are economically and practically infeasible. We show that, given a small amount of user interaction (little more than that required to build a conventional morphable model), there is enough information in a collection of 2D pictures of certain object classes to generate a full 3D morphable model, even in the absence of surface texture. The key restriction is that the object class should not be strongly articulated, and that a very rough rigid model should be provided as an initial estimate of the 'mean shape'.

The model representation is a linear combination of subdivision surfaces, which we fit to image silhouettes and any identifiable key points using a novel combined continuous-discrete optimization strategy. Results are demonstrated on several natural object classes, and show that models of rather high quality can be obtained from this limited information.

Index Terms—Morphable model, shape from silhouette, subdivision surfaces, image-based modelling, single-view reconstruction.

1 Introduction

Morphable models of the human head and body have enabled a myriad of applications in computer graphics, special effects, and computational photography [1], [2], [3], [4], [5]. However, building a morphable model for a new object category is currently a task that demands considerable effort: even if full 3D scans of many object instances are available, existing systems require user-guided correspondence algorithms to build a vertex-consistent mesh across instances. For some classes, such as live animals, the difficulty is even greater: obtaining the 3D scans is essentially impossible even using today's real-time depth sensors, particularly if canonical poses of the animals are dynamic, for example a dolphin in mid-leap. In one sense, however, we are not short of data: we can obtain dozens of high-quality images of the target class simply using a search engine, much as Photo tourism [6] makes use of images of tourist landmarks. The question we ask in this paper is whether such a collection contains enough information to build a 3D morphable model of a new object class. For instance, given 32 images of the class 'dolphin', can we generate an 8-parameter morphable model which fits the image data, and generates plausible new 3D dolphin instances? We show that with a small amount of additional information supplied by user interaction and a rough initial estimate of the 'mean shape', morphable models of usable quality can be obtained purely from such 2D images (see Figure 1). The models we obtain are at a coarser scale than those from 3D scans, but nevertheless capture the object class in a manner that has not previously been possible. The classes to which this applies are general nonrigid or deformable objects without significant articulation (e.g. pigeons without feet, but not the human hand). Importantly, surface texture is not required: the major source of information is the object's silhouette.

The contributions of this work are at a number of levels.

- The first is essentially scientific: showing that a 3D object *class* model can be recovered from 2D images is an extension of existing results in nonrigid structure from motion [7], in multiview reconstruction from silhouettes [8], [9], and in single-view reconstruction [10].
- The second level is technical: a new combined continuous-discrete optimization for recovery of the object shape and contour generator (§5), and new constructions for optimization on subdivision surfaces (the details of which appear in supplementary material).
- Finally, the practical contribution is to enable the many applications to which morphable models have been put to be extended to other classes.

1.1 Related work

Cootes et al. [11] build the first class-specific morphable models of 2D shapes from 2D contours. By analysing the relationship between linear combinations of 3D models and their 2D projections, Vetter and Poggio [12]

- *T. Cashman is with the Faculty of Informatics, University of Lugano, Switzerland. E-mail: thomas.cashman@usi.ch.*
- *A. Fitzgibbon is with Microsoft Research, 7 JJ Thomson Ave, Cambridge, UK. E-mail: awf@microsoft.com.*

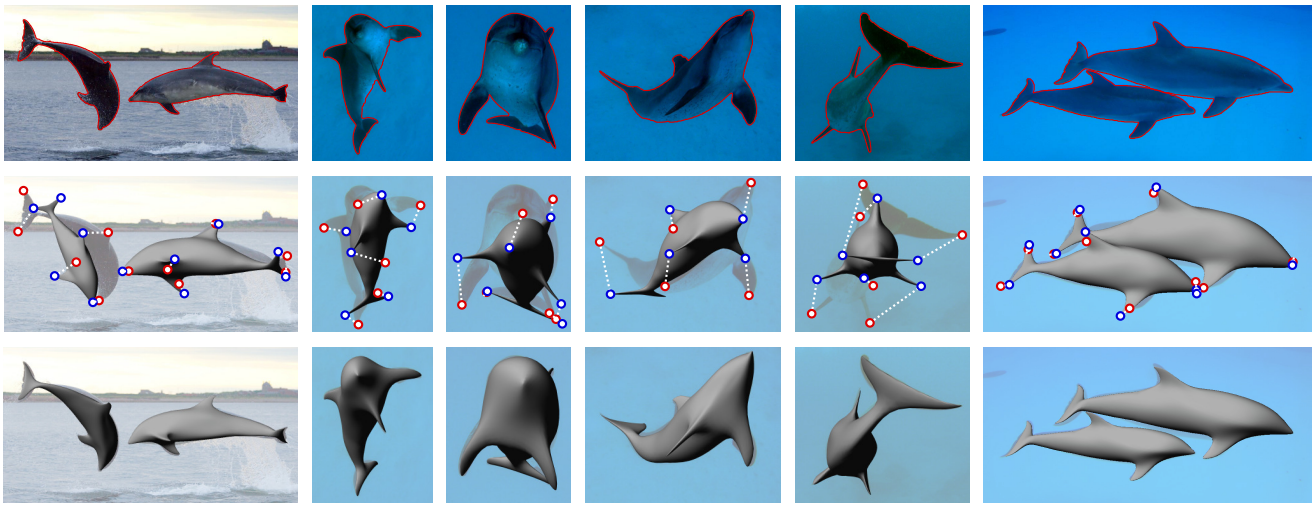


Fig. 1. Images showing eight dolphins out of 32, from which we build an 8-parameter morphable model. Top: input images with silhouette annotations. Middle: The rigid dolphin prototype in initial position for our optimization, showing user-provided point constraints (blue), each of which corresponds to a point in the image (shown in red). Bottom: final morphable model reconstruction overlaid on the input images.

indicate the potential for 'linear object classes' to describe a variety of complex phenomena. Applied to modelling of the human head from 3D scans, Blanz and Vetter [1] demonstrate the enormous power of 3D morphable models fitted to 2D images; extended to modelling of the whole body, the range of applications continues to widen [5]. Chen et al. [13] learn separate pose and shape models for sharks as well as humans, and demonstrate good reconstructions by fitting these models to a single silhouette. However, in all of these cases, 3D models are built starting from 3D data, while we wish to recover 3D shape from 2D images.

1.1.1 Single-view reconstruction

The recovery of 3D from 2D is a subject with a long history. The first class of related techniques comprises those which recover a 3D model from a single 2D image. Terzopolous et al. [14] show how generalized cylinders can be fitted to image silhouettes using an iterative approach which may be viewed as energy minimization. Gingold et al. [15] combine generalized cylinders with user-provided semantic annotations to create smooth models from 2D sketches. Using simple image-based rules, Igarashi et al. [16] generate a variety of impressive 3D models in the TEDDY system. Karpenko and Hughes [17] allow more topologies in SMOOTHSKETCH and deal with invisible cusps and incomplete silhouettes. Prasad et al. [10] show how geometry images [18] allow complex topologies to be recovered using a globally-convergent quadratic energy minimization. The FiberMesh system [19] unifies several existing constraint sources in an energy minimization framework to provide a comprehensive interactive system for 3D model construction from curves.

Our approach may be seen as a generalization of this type of system to morphable models rather than a single 3D model. However, although previous systems could use a fixed assignment of the contour generator's preimage in surface parameter space with little effect on the final model, this approach is inadequate for the morphable model case. Kraevoy et al. [20] find contour generator preimages using a dynamic programming approach similar to the discrete optimization we propose in §5.2. They do not combine this with a continuous representation, however, and their focus is fitting a single 3D mesh to a single sketched contour.

If we consider how these approaches might be adapted to the problem we address in this paper, one might imagine performing a single-view reconstruction for each of our input images and then applying existing morphable model fitting methods. However, this approach would ignore the shape information available from multiple viewpoints represented across the images, depending heavily on the surface smoothness priors used in each reconstruction, as well as requiring considerably more effort than our proposal.

1.1.2 Rigid reconstruction from multiple views

A second school of 3D recovery uses the information available in multiple images of a single rigid 3D object or scene. For textured objects, very effective systems exist based on interest point detection and matching. For example, Photo tourism [6] shows that 3D models can be built from internet-sourced images. However, our system cannot depend on interest points alone, for two reasons: first, many objects of interest are untextured (e.g. many animals); second, even textured object classes such as leopards or giraffes may not have *corresponding* interest points across individuals. Recall that each of our images may be of a different individual.

The key prior work in rigid reconstruction is therefore systems which recover shape and motion from contours, and in particular from the object's occluding contour or silhouette. With known camera motion,

shape from silhouette is a relatively straightforward problem with a rich literature. Thus the key problem is to recover camera motion from silhouette information, which is difficult to do effectively. For the special case of two images of a single rigid object, Porrill and Pollard [21] show that "frontier points"—the 3D points where epipolar planes touch the object—can provide point correspondences, and Cipolla et al. [22] show how the frontier points can constrain relative camera motion. Furukawa et al. [8] show that camera positions can be computed for multiple images using the frontier points from all pairs of images, combined with some contour tangent information. McIlroy and Drummond [9] demonstrate camera recovery in the same scenario without the tangency information. For our purposes, however, these solutions share a significant disadvantage: frontier points are very much a property of the same rigid object seen in two cameras. As each of our images views a different 3D object, there is no analogue of the frontier point, so the above methods cannot be applied. However, even without explicit use of frontier points, one contribution of our work is to compute camera positions for multiple cameras while simultaneously recovering object structure all along the silhouette.

1.1.3 Nonrigid reconstruction from multiple views

Moving from rigid to nonrigid reconstruction, almost all existing work is based on interest points tracked through video. In a sequence of papers, Torresani and collaborators [23], [7] extend rigid-geometry structure from motion (SfM) to nonrigid scenes using the same idea as in morphable model construction: the parameters of the model in a given image are assumed to be representable as a linear combination of the parameters of a set of 3D basis models. However, as with rigid SfM, the input is the same 3D object, so point correspondences can be obtained, unlike in our scenario. Recently, Prasad et al. [24] addressed this problem by using curve features rather than feature points, arguing that for some object classes, there are 3D curves in common across object instances. For example, plant leaves and petals can be modelled by the 3D curves comprising their outer boundaries; or surface markings on some animal species are common across members of the class. However, suitable object classes are relatively few, while the technique we describe in this paper applies to a much larger set of shape classes, because the object silhouette provides such strong shape cues and point constraints can be included.

Perhaps the closest work to ours is that of Ilić et al. [25], who fit deformable 3D models to video using a combination of silhouette and point features. The key differences are that the number of point correspondences they use is significantly higher than in this work; they find silhouettes by solving an ODE at each iteration, rather than via the global search we employ in §5.2; the deformations they consider are quite small, and in each of their examples, recovery is aided by some extra information such as a predefined morphable model (in contrast, we initialize with a predefined *rigid* model) or a deformation basis from eigenshapes (with unsatisfactory results as shown in §6.4 below).

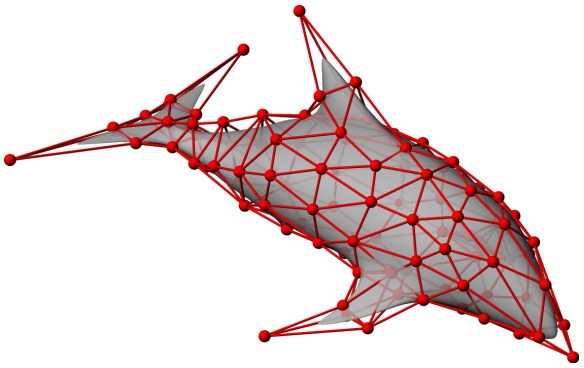


Fig. 2. A rigid template triangulation (red lines) which we use to initialize our dolphin model. Note that this triangulation defines only the mesh topology, and is defined just once per object class, *not* per image. The Loop subdivision surface defined by combining this triangulation with the vertex positions defined by the red spheres is shown in gray.

2 Morphable Model

In previous work on 3D morphable models, the model is represented by a mesh of control vertices, combined with an interpolation rule to create a continuous surface from the mesh. For example, triangular facets [1], [5] yield a piecewise planar model. Such models are reasonably easy to use, but are described by thousands of parameters, which would make them extremely unwieldy in our application. In contrast, we assume piecewise smooth objects, and use subdivision surfaces (see §5.4), which require many fewer parameters to describe, at the cost of a more complex interpolation rule. Despite its apparent complexity, the model surface is still linear in the control vertices.

The use of subdivision surfaces does impose the requirement that the model topology must be predefined, which is most easily achieved by defining a rigid model of approximately the right shape (see Figure 2). We created our primary template meshes in the 3D modeller Blender, but also tested models generated with even less user interaction by using mesh decimation methods [26] on the output of the sketch-based modeller FiberMesh [19]. See §6.1 for details of this process, where we also demonstrate that the template does not need to be accurate in terms of shape, and simply defines the mesh topology.

We defer the precise details of the model until §5.4, and specify only that both our model and the previous mesh-based models may be given as follows: we have an explicit surface representation parametrized by p control vertices represented by a vector $\boldsymbol{X} \in \mathbb{R}^{3p}$ and

evaluated on a 2D domain Ω. Initially it will suffice to think of Ω as a subset of $\mathbb{R}^2$, but its topology is in practice much less restrictive. Points in Ω will be denoted using a circle accent, for example $\mathring{u}$. The surface is a mapping $M : \Omega \mapsto \mathbb{R}^3$, where each point $\mathring{u}$ generates a 3D surface point denoted by $M(\mathring{u}|\boldsymbol{X}) \in \mathbb{R}^3$, to make explicit the dependence on the control vertices $\boldsymbol{X}$. $M(\mathring{u}|\boldsymbol{X})$ is linear in $\boldsymbol{X}$ but non-linear in $\mathring{u}$; for our choice of Loop subdivision (see §5.4), M is actually a piecewise quartic polynomial function of $\mathring{u}$. The corresponding surface normal is written $N(\mathring{u}|\boldsymbol{X}) \in \mathbb{R}^3$.

Our model class allows for sharp edges, and indeed perfect sharp edges can be generated by tagging certain model edges as crease edges. In practice, this adds complexity that we have not implemented, but as the dolphin results show, good results can be obtained simply by allowing the control mesh vertices to be near each other.

Let $\boldsymbol{X}_i$ be the control vertices for the surface describing the object in image i. Then we define our morphable model as a linear combination of $D+1$ basis shapes, denoted $\boldsymbol{B}_m \in \mathbb{R}^{3p}$, giving

$$\boldsymbol{X}_i = \sum_{m=0}^{D} \alpha_{im} \boldsymbol{B}_m \quad \text{with} \quad \alpha_{i0} = 1. \tag{1}$$

From the constraint that $\alpha_{i0} = 1$, we expect $\boldsymbol{B}_0$ to represent an instance of the target object in neutral pose, and the other basis shapes to encode the pose and shape variations required to explain the image data. This is analogous to performing principal component analysis (PCA) on a set of known shapes [1], [5], where $\boldsymbol{B}_0$ plays the role of the data mean, and the other basis shapes $\{\boldsymbol{B}_m\}_{m=1}^{D}$ correspond to the PCA modes of variation. However since the shapes $\{\boldsymbol{X}_i\}_{i=1}^{n}$ are *unknown* a priori, this analogy to PCA is not exact: we do not constrain the basis shapes $\{\boldsymbol{B}_m\}_{m=1}^{D}$ to be orthogonal to one another, and neither is $\boldsymbol{B}_0$ equal to the mean $\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{X}_i$. Nevertheless this model does give a D-dimensional morphable model (i.e. the same expressive power as PCA with D modes of variation), as long as the shapes $\{\boldsymbol{B}_m\}_{m=1}^{D}$ are independent. We use an iterative relaxation (§5.3) to guide the model towards such independent shapes.

3 Inputs

Before developing the details of our method, it will be useful to look at the inputs it requires. Some of these inputs are currently supplied by user interaction, allowing us to ensure that the input does not appear as a significant source of error in our results (§6). The interactions are designed to be quick and easy for the user, and in many cases, the input could also be made automatic, as researchers have demonstrated automatic systems which provide similar data. We make it clear where this applies in the sections below.

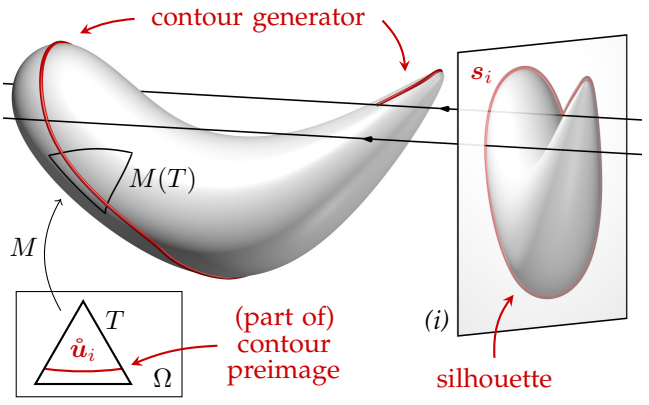


Fig. 3. An image *(i)* of an object with the silhouette $\boldsymbol{s}_i = \{\boldsymbol{s}_{ij}\}_{j=1}^{S_i}$. The contour generator projects to this silhouette in the image. We also show one triangle $T \subset \Omega$, and a part of the contour preimage $\mathring{\boldsymbol{u}}_i$ that appears in T. The contour generator (which is *discontinuous*, in this figure) is the result of using M to evaluate the surface at the contour preimage.

The main input (apart from the rigid template described above) is a collection of images containing n instances of the object class of which we wish to build a model. We primarily use the example of the class of dolphins, as they exhibit reasonably complex deformations, and the combination of smooth and sharp edges represents a challenge for modelling. For the remainder of the paper, we assume that each image contains exactly one object instance, so images containing more than one are simply indexed twice. We index per-image quantities using i ranging from 1 to n.

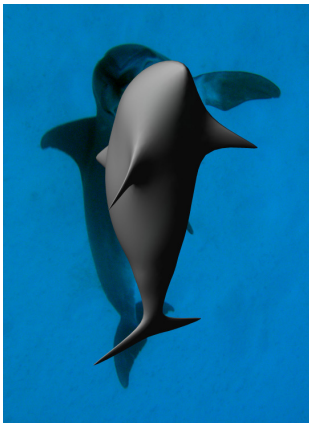
3.1 Image data: Silhouette

The primary feature in our approach is the object's *silhouette*, or *occluding contour*. Figure 3 illustrates some of the key terms we use in this paper. The silhouette is a 2D curve in the image plane, which is the projection of a possibly discontinuous 3D space curve called the *contour generator*. The contour generator is a curve on the object surface, and thus is associated with a curve *in the parameter space* Ω. We call this parameter-space curve the *contour preimage*.

We extract the silhouette of each object instance using a closed curve (see Figure 4b). This could also be made automatic by training a 2D class model [27]. Each silhouette is sampled to create S_i discrete silhouette points $\boldsymbol{s}_{ij} \in \mathbb{R}^2$ $(j = 1..S_i)$, each of which has a corresponding unit normal $\boldsymbol{n}_{ij} \in \mathbb{R}^2$ (see Figure 4d). The silhouette as a whole in image i is written

$$\boldsymbol{s}_i = \{\boldsymbol{s}_{ij}\}_{j=1}^{S_i}.$$

Note that some definitions of image silhouette include internal edges. These are not required in our framework—see for example Figure 4b, where the right fin is only partially represented on the image silhouette.



(a) The initial dolphin model (see Figure 2) overlaid in the position given by the initial camera view $\{\boldsymbol{t}_i, \sigma_i, \boldsymbol{R}_i\}$ for this image.

- basis shapes $\{\boldsymbol{B}_m\}_{m=0}^{D}$,

and per-image

- shape coefficients $\{\boldsymbol{\alpha}_i\}_{i=1}^{n}$ where $\boldsymbol{\alpha}_i := \{\alpha_{im}\}_{m=1}^{D}$,
- camera parameters $\{\boldsymbol{t}_i, \lambda_i, \boldsymbol{R}_i\}_{i=1}^{n}$,
- contour preimages $\{\mathring{\boldsymbol{u}}_i\}_{i=1}^{n}$, where $\mathring{\boldsymbol{u}}_i := \{\mathring{u}_{ij}\}_{j=1}^{S_i}$.

We gather the shape and pose parameters into a set $P = \{\boldsymbol{B}_0 \,..\, \boldsymbol{B}_D, \boldsymbol{\alpha}_1 \,..\, \boldsymbol{\alpha}_n, \boldsymbol{t}_1 \,..\, \boldsymbol{t}_n, \lambda_1 \,..\, \lambda_n, \boldsymbol{R}_1 \,..\, \boldsymbol{R}_n\}$, and the preimage into $U = \{\mathring{\boldsymbol{u}}_1 \,..\, \mathring{\boldsymbol{u}}_n\}$. Although this sea of symbols represents a large number of parameters, the silhouette is such a rich source of shape information that the number of measurements greatly outweighs the number of unknowns, so we can expect a relatively well-constrained solution. We provide a glossary of our notation in the supplementary material.

4.1 Matching image silhouettes and constraints

To begin, we shall assume we already have the contour preimage points $\mathring{u}_{ij}$, each of which corresponds to a silhouette sample $\boldsymbol{s}_{ij}$. Then the primary goal of our surface reconstruction is that evaluating the model at the contour preimage $\mathring{\boldsymbol{u}}_i$, and viewing from the correct angle, should give the silhouette $\boldsymbol{s}_i$. That is, with a perfect model in the absence of noise, the reconstruction would satisfy

$$\boldsymbol{s}_{ij} = \pi_i\left(M(\mathring{u}_{ij}|\boldsymbol{X}_i)\right) \qquad \forall i, j. \tag{4}$$

Our first energy term is the negative log probability of deviation from this equality under a Gaussian distribution of variance σ_{sil}^2:

$$E_i^{\text{sil}} = \frac{1}{2}\sigma_{\text{sil}}^{-2}\sum_{j=1}^{S_i}\left\|\boldsymbol{s}_{ij} - \pi_i\left(M(\mathring{u}_{ij}|\boldsymbol{X}_i)\right)\right\|^2, \tag{5}$$

where σ_{sil} is an estimate of the noise in the extracted silhouettes. There is a similar energy term for the user-specified constraints; compare with (2):

$$E_i^{\text{con}} = \frac{1}{2}\sigma_{\text{con}}^{-2}\sum_{k=1}^{K_i}\left\|\boldsymbol{c}_{ik} - \pi_i\left(M(\mathring{\mu}_{ik}|\boldsymbol{X}_i)\right)\right\|^2. \tag{6}$$

For the surface to be consistent with the silhouette at $\boldsymbol{s}_{ij}$, we require the normal $N(\mathring{u}_{ij}|\boldsymbol{X}_i)$, once transformed by $(\boldsymbol{R}_i^{-1})^T = \boldsymbol{R}_i$, to lie in the plane spanned by $[\boldsymbol{n}_{ij}^T\ 0]$ and $[0\ 0\ 1]$. However, we also want the reconstructed surface to generate a silhouette at the contour generator, which means that at $\mathring{u}_{ij}$ the surface must be normal to the viewing direction. Prasad et al. [10] show that we therefore know the required normals completely: in the image space which is the target of $\boldsymbol{R}_i$, we want the normal to be exactly $[\boldsymbol{n}_{ij}^T\ 0]$. This leads to the energy term

$$E_i^{\text{norm}} = \frac{1}{2}\sigma_{\text{norm}}^{-2}\sum_{j=1}^{S_i}\left\|\begin{bmatrix}\boldsymbol{n}_{ij}\\ 0\end{bmatrix} - \nu\left(\boldsymbol{R}_i N(\mathring{u}_{ij}|\boldsymbol{X}_i)\right)\right\|^2 \tag{7}$$

where ν is the normalization function $\nu(\boldsymbol{x}) = \boldsymbol{x}/\|\boldsymbol{x}\|$.

These terms penalize solutions where the surface at the contour generator is not consistent with the observed image silhouette. However they do not penalize 'spillage', where an unconstrained part of the surface falls outside of the observed image silhouette when projected into the image plane. We discuss the modifications needed to penalize spillage in §7. Nevertheless we find that the constraints imposed on the solution by E_i^{norm} are sufficient for many object classes in practice (see Figure 1).

4.2 Smoothness and regularization

At the contour generator curves, we have a rich set of information on the surface shape from the silhouettes. Away from these curves, however, the surface is far less constrained. In order to find an acceptable solution, we therefore have to incorporate prior knowledge on the types of surfaces that we wish to reconstruct. Here we make the assumption that the target surfaces are smooth, and we can choose from the wide array of available surface smoothness energies [29].

One of the simplest choices is the linearized thin-plate energy, and this has a form that can be evaluated exactly and efficiently [30] for our surface representation, subdivision surfaces. A disadvantage is that the thin-plate energy is a poor fairness measure for surfaces which are given by only planar (or near-planar) boundary constraints [19]. This is the case when reconstructing surfaces from a single view. However, we have the advantage that our solution is constrained by multiple views by means of the morphable model (1). We can therefore exploit the simplicity of the thin-plate energy without suffering from flat reconstructions.

There is a corresponding energy term for each of the $D+1$ basis shapes. Using subscripts to indicate partial differentiation in orthogonal parametric directions x and y, these terms are:

$$E_m^{\text{tp}} = \frac{\bar{\lambda}^2}{2}\int_\Omega \|M_{xx}(\mathring{u}|\boldsymbol{B}_m)\|^2 + 2\,\|M_{xy}(\mathring{u}|\boldsymbol{B}_m)\|^2 + \|M_{yy}(\mathring{u}|\boldsymbol{B}_m)\|^2\,\mathrm{d}\mathring{u}. \tag{8}$$

The thin-plate energy is not invariant to scaling of the basis shapes $\boldsymbol{B}_m$, so we weight these smoothness terms by the square of the average orthographic scale factor, $\bar{\lambda} = (\sum_i \lambda_i)/n$, to avoid a solution with arbitrarily large values of λ and correspondingly small $\boldsymbol{B}_m$. Similarly, we need to regularize the shape coefficients α_{im}, as otherwise the optimization will move towards a solution with small $\boldsymbol{B}_m$ where $m > 0$. We therefore add the regularization terms

$$E_i^{\text{reg}} = \beta\sum_{m=1}^{D}\alpha_{im}^2. \tag{9}$$

Here the value of the weight β simply fixes the otherwise unconstrained overall scale of the deformation basis shapes, and can be set to any value.

4.3 Contour generator continuity

For simple scenes we expect the contour generator to be a continuous curve, and we want to prevent a result where the discrete points that we use to represent the contour preimage are arbitrarily scattered over Ω. We therefore add a term that penalizes a large distance between points in the contour preimage. Conceptually we can write that distance as $\|\mathring{u}_{i,j+1} - \mathring{u}_{ij}\|$, although our use of subdivision surfaces means that Ω has a piecewise structure. We therefore approximate the distance between points in Ω by a function $d(\mathring{u}_{ij}, \mathring{u}_{i,j+1})$ which takes account of the structure; see the supplementary material for details.

Although silhouettes are mostly the image of *continuous* curves on the surface, we also need to allow for selected *discontinuities* where one part of a surface obscures or appears behind another (see Figure 3). Such discontinuities should incur only a fixed penalty, unrelated to the size of the jump on the surface. We therefore add the distance function d to the energy using the truncated quadratic $\tau(x) = \min(x^2, h^2)$ for a problem-independent constant h (see supplementary material), giving

$$E_i^{\text{cg}} = \gamma \sum_{j=1}^{S_i} \tau(d(\mathring{u}_{ij}, \mathring{u}_{i,j+1})) \tag{10}$$

where γ is a weighting parameter which is required to find good solutions for contour generators, but should be small and need not change between object classes; in fact γ could be relaxed to 0 by the end of the optimization.

4.4 Complete energy function

In summary, by combining all of the terms (5) to (10), we arrive at the following energy

$$\begin{aligned} E = &\sum_{i=1}^{n} \left(E_i^{\text{sil}} + E_i^{\text{norm}} + E_i^{\text{con}} \right) + \\ &+ \sum_{i=1}^{n} \left(E_i^{\text{cg}} + E_i^{\text{reg}} \right) + \xi_0^2 E_0^{\text{tp}} + \xi_{\text{def}}^2 \sum_{m=1}^{D} E_m^{\text{tp}}. \end{aligned} \tag{11}$$

We weight the smoothness terms E_0^{tp} and E_m^{tp} (for $m > 0$) separately, as this allows us to distinguish between the smoothness of the target object class and the smoothness of the model, i.e. how smoothly the model deforms when there is a change in the shape coefficients α. See Figure 11 for a demonstration of the effect of these separate weights.

Now that we have defined E, our task is to find its minimum under perturbation of the parameters defined at the start of this section. This gives a highly nonlinear and large-scale optimization problem, which implies that we need a good initialization (since the search space may contain many local minima), and an optimization algorithm specific to this problem domain (in order to make the search tractable). The next section describes our strategy to address both of these challenges.

5 Optimization algorithm

The energy E is a function of pose and shape parameters P and of the preimage parameters U, which may be viewed as 'nuisance' or latent parameters. Thus our optimization problem may be written as

$$P^* = \operatorname*{argmin}_P \min_U E(P, U).$$

As both sets of parameters are continuous, a general smooth-function optimizer may be used to simultaneously optimize $E(P, U)$, computing

$$(P^*, U^*) = \operatorname*{argmin}_{P,\,U} E(P, U)$$

and discarding U^*. By simultaneously optimizing over P and U, we can rapidly converge to a nearby local minimum, and this forms one component of our overall optimization strategy. However it is difficult to find a good initial estimate for U, so this strategy does not find high-quality optima (see Figure 7b).

We therefore interleave this continuous optimization stage with a global discrete optimization over the U only. Thus the overall strategy, given (P^m, U^m), is to compute (P^{m+1}, U^{m+1}) by iterating the two steps

$$U^* = \operatorname*{argmin}_{U \in \Phi^S} E(P^m, U), \tag{12}$$

using global optimization over the discrete set Φ (see §5.2), and

$$(P^{m+1}, U^{m+1}) = \operatorname*{argmin}_{P,\,U} E(P, U), \tag{13}$$

using local continuous optimization starting from (P^m, U^*) (see §5.1).

5.1 Local non-linear optimization

For local continuous optimization, we use the trust-region-reflective algorithm provided by the MATLAB function `lsqnonlin`. The key observation that allows us to use a least squares minimization is that each term of E can be written as a squared quantity. To see this for E^{tp}, observe that $M(\mathring{u}|\boldsymbol{B}_m)$ is linear, and hence E_m^{tp} is quadratic, in $\boldsymbol{B}_m$. We can therefore write E_m^{tp} as a quadratic form $\boldsymbol{B}_m^T \boldsymbol{Q} \boldsymbol{B}_m$ for a coefficient matrix $\boldsymbol{Q}$ which is positive definite. By taking the matrix square root of $\boldsymbol{Q}$, we arrive at a vector $\boldsymbol{w}_m = \boldsymbol{Q}^{1/2} \boldsymbol{B}_m$ such that $E_m^{\text{tp}} = \|\boldsymbol{w}_m\|^2$.

Each camera is given the 6-parameter representation described in §3.3. To represent the rotation, we use $\exp([\boldsymbol{\theta}_i]_\times)$, the matrix exponential of the skew-symmetric matrix $[\boldsymbol{\theta}_i]_\times$ with values drawn from the vector $\boldsymbol{\theta}_i \in \mathbb{R}^3$. The exponential map has singularities wherever $\|\boldsymbol{\theta}_i\|$ is a multiple of 2π [31]. To avoid these, we combine the optimized rotation with the user-specified initial rotation $\hat{\boldsymbol{R}}_i$, by setting $\boldsymbol{R}_i = \hat{\boldsymbol{R}}_i \exp([\boldsymbol{\theta}_i]_\times)$. This keeps $\|\boldsymbol{\theta}_i\|$ small by using the optimized rotation as a small correction to $\hat{\boldsymbol{R}}_i$.

5.2 Global search for contour generators

E is quadratic in α_i and $\boldsymbol{B}_m$, but has a much more complicated dependence on the camera parameters and contour preimages. The dependence on the camera parameters is quite well understood in the context of bundle adjustment [32], and here we are also able to take advantage of a good initial estimate (§3.3). However, the contour preimages are still very susceptible to local minima, and their convergence has a large effect on the quality of the final solution.

We therefore interleave iterations of least squares minimization with a global search for contour generators, by selecting preimage points from a set of discrete candidate positions $\Phi \subset \Omega$. Our experiments use $10g + p$ candidates: 10 sampled uniformly from inside each of the g triangles in the control mesh, and one corresponding to each of the p control vertices. However the density of this sampling is not critical, as the continuous optimization follows every global search and finds the optimal local position for each preimage point.

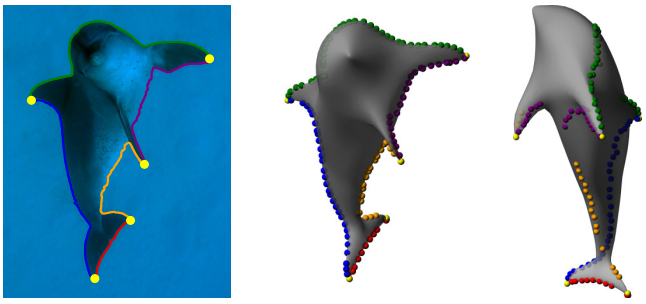
As the global search modifies only the contour preimage points U, the terms E^{con}, E^{tp} and E^{reg} are all constant. Each image therefore presents a separate optimization problem, and finding the minimum of E with respect to the contour preimage path for image i coincides with finding

$$\min_{\mathring{\boldsymbol{u}}_i} \left(E_i^{\text{sil}} + E_i^{\text{norm}} + E_i^{\text{cg}} \right). \tag{14}$$

This can be solved efficiently using dynamic programming, where subproblems solve for a shorter contour of length l. The base case, where $l = 1$, evaluates only E^{sil} and E^{norm} for each of the candidate positions in Φ. Adding an additional point, to increase l by one, uses the penalty E^{cg} in considering each possible transition between preimage points. Once l has reached the required contour length, we can find the optimal preimage path by tracing back through the calculated cost matrix.

We are often able to accelerate the calculation of the globally optimal path even further, by noting any constraints $\boldsymbol{c}_{ik}$ which lie on the silhouette. In Figure 4, for example, there are five such points. Since these constraints associate a $\mu_{ik} \in \Omega$ with a point on the silhouette, they partition the contour path problem into separate sections, each of which can be solved separately (see Figure 5).

In the unusual situation where no constraints lie on the silhouette, we do not have separate linear path problems but just one *circular* path to find. In this case, we have to make sure that the path forms a closed loop, since dynamic programming makes no such guarantee if used alone. A naive solution would be to solve equation (14) $|\Phi|$ times, each time constraining a linear path to start and end at a given candidate point. However, Appleton and Sun [33] show that the same result can be found with logarithmic rather than linear complexity, by using a branch-and-bound method. Using their algorithm, we solve a linear dynamic programming problem at each iteration, where the path is constrained to start and end in some subset of Φ. The resulting path is a lower bound for the energy of a circular path that starts and ends in the given subset, and if the resulting path is circular, then it is also a possible solution to the problem. The algorithm continues pruning a binary search tree for subsets of Φ until there is a circular solution, and no other subset has a smaller lower bound.



(a) Silhouette, with constraints $\boldsymbol{c}_{ik}$ that lie on the silhouette marked in yellow (see Fig. 4b).

(b) Two different views showing the result of solving (14) separately on each coloured section, between the known points $M(\mu_{ik})$ marked in yellow.

Fig. 5. If one or more constraints $\boldsymbol{c}_{ik}$ lie on the silhouette, the global search for contour generators is partitioned into separate problems for non-circular paths. In this example, the constraints shown in (a) split the silhouette into the five sections shown in different colours, which we solve in turn to gain the contour generator solutions in (b).

For each of the eight dolphins shown in Figure 1, a naive search for an initial circular contour generator path takes one hour. Appleton and Sun's algorithm finds the same globally optimal circular paths in an average of 62 seconds, with solution times ranging from 29 to 138 seconds depending on the number of passes required to prune the binary search tree. Finally, by splitting each contour generator problem into separate linear path problems using point constraints that lie on the silhouette, each problem can be solved in less than 2.5 seconds. With these improvements, runtime is therefore dominated by the continuous least-squares minimization described in §5.1.

5.3 Initialization and optimization schedule

We initialize $\boldsymbol{B}_0$ with the control vertices of the template, and the initial camera rotation with user input as described in §3: this gives the initialization P^0 for the pose and shape parameters P. The first run of our global search (12) provides an initial estimate U^* for the contour preimages $\mathring{\boldsymbol{u}}_i$. In total we use D passes each of the global discrete optimization (12) and the local continuous optimization (13), adding one additional basis shape for each pass. The final output of our algorithm is then (P^D, U^D), from which we can discard the preimage parameters U^D. At each pass, we initialize the newly-added $\boldsymbol{B}_m$ to $\mathbf{0}$ and α_{im} to 1 for all i.

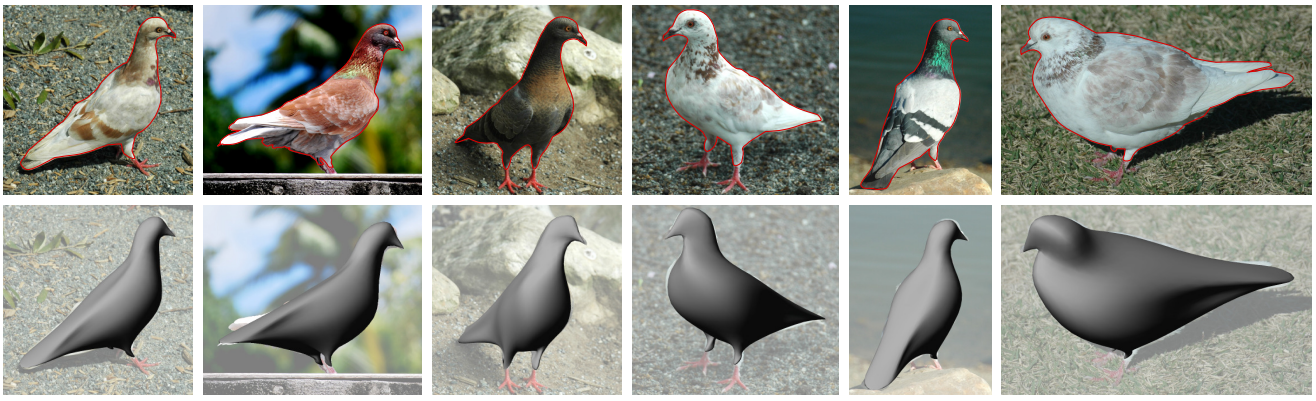


Fig. 6. Images showing six pigeons out of 25, from which we build a 7-parameter morphable model. Top: input images with silhouette annotations. Bottom: final morphable model reconstruction overlaid on the input images.

Gradually increasing the dimension of the model is more reliable than using D passes which optimize for all $D+1$ basis shapes at once, as the model is able to successively factor out increasing amounts of variation in pose and shape. The solution is therefore directed towards a result with independent shapes $\boldsymbol{B}_m$, although explicit orthogonality is not imposed (nor is it necessary).

5.4 Subdivision surface model

To minimize the energy E given in §4, we need to define the functions M and N. We use Loop subdivision surfaces [34], which have several advantages for this application. Loop surfaces are linear functions of the control vertices $\boldsymbol{X}$, maintain continuity as $\boldsymbol{X}$ is modified, and can be evaluated at arbitrary positions in the domain Ω [35]. Surface normals are well defined everywhere, and the same is true of their derivatives except at isolated singularities. A smooth surface representation is particularly important given our continuous search for contour preimages $\mathring{\boldsymbol{u}}_i$, which relies on the continuity of the terms E_i^{norm}, which in turn are functions of the surface normal.There is also a closed form for the linearized version of thin-plate energy integrated over the parametric domain [30], leading to an efficient evaluation of our terms E_m^{tp}.

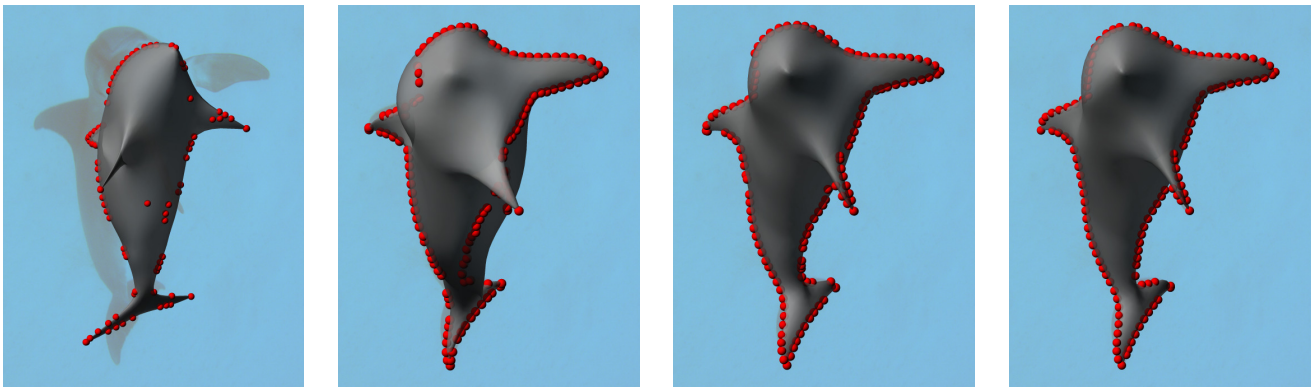
However, the piecewise structure of the parameter space Ω presents two main challenges: in order to evaluate E^{cg}, we need a function $d(\mathring{u}_j, \mathring{u}_{j+1})$ which serves as a distance measure between positions in Ω, and we need to define how differential updates $\frac{\partial E}{\partial U}$ are executed on the piecewise domain. We address both problems by locally reparametrizing Ω about the current estimate, following ideas developed by Peters, Reif [36] and Zorin [37]. The details are provided in the supplementary material, but Figure 7 gives an intuitive indication of the benefit of this reparametrization.

6 Experiments

We implemented the system as described, and tested it on four object categories. For all experiments we set $S_i = 125$ for all i, and used $\gamma = 1/128$ and $\beta = 0.5$ for all problems. The silhouette point noise level $\sigma_{\text{sil}} = 1$ (the units are arbitrary as an overall scale may be applied to the energy), and the point constraint noise level $\sigma_{\text{con}} = 0.2$. σ_{norm} is set on a per-problem basis below as the noise on normals depends on the silhouette extraction process. We empirically determine the shape parameters $\{\xi_0, \xi_{\text{def}}\}$ by modifying default values based on a visual assessment of reconstruction quality; see §6.2 for an experiment that measures the sensitivity of our system to ξ_0 and ξ_{def}. We ran all the experiments on a laptop with an Intel Core i7 processor clocked at 2.67 GHz. The source code for our implementation is available from http://forms.codeplex.com/.

The first dataset contains 32 dolphin instances in a variety of poses: 22 taken underwater, and 10 showing dolphins in the middle of jumps out of the water. We extracted silhouettes using the 'Remove Background' feature of PowerPoint 2010. For 17 images this required no more than a bounding box, but for the rest, additional mouse strokes were required. On average, 1.63 extra mouse strokes were required per image. The weights we use for this dataset are $\sigma_{\text{norm}} = 1/0.075 \approx 13.3$, $\xi_0 = 0.5$ and $\xi_{\text{def}} = 0.25$. Our optimization finds a morphable model with 8 parameters in 24 hours. The number of variables involved in each iteration of the local non-linear optimization ranges from 8,267 (when $D = 1$) to 10,584 (when $D = 8$). See Figure 1 and the accompanying video for examples of reconstructions from this model.

We used this dataset to test the impact of varying the number of images used to build a model. Figure 8 shows that with a small number of images, such as the 8 used to reconstruct Figure 8 (left), the contour generators may not sufficiently cover the model surface. The result is a reconstruction which is too flat, given this disadvantage of the thin-plate energy we mentioned in §4.2. However, 16 images is enough to gain a plausible reconstruction, for this example, and using the full 32 images allows the model to return a more refined pose estimate.



(a) Initial estimate. (b) Only continuous local optimization, as described in §5.1. (c) As (b), but including iterations of our global search (§5.2). (d) As (c), but with reparametrization around singularities.

Fig. 7. Different optimization methods for learning a morphable model with 8 parameters from 32 dolphin instances. In each image, the final solution surface is shown transparent and overlaid with the contour generator shown as red spheres. In (b), the preimage points belonging on the dolphin's foremost fin are stuck in local minima. This deficiency is repaired in (c), but poor parameter updates damage the convergence of the contour preimage points. In (d) the optimizer finds a good solution for nearly all preimage points.

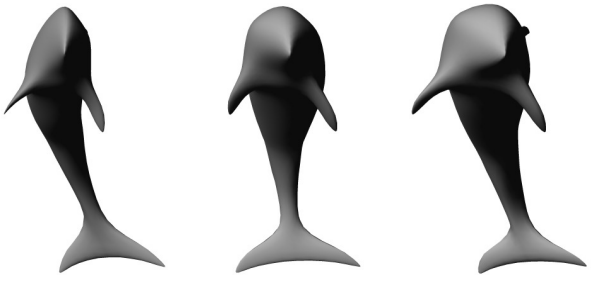


Fig. 8. A reconstruction of the same dolphin, using 8, 16, and 32 instances (from left to right) to build an 8-parameter model.

The second dataset was built from images of 25 pigeons, selecting images where the birds' wings are folded rather than spread. We chose images which show an interesting variety of poses for the head, and also a large spread of different types of pigeons. To simplify the problem and avoid occlusion, we modelled only the feather-covered part of each bird, excluding the featherless feet in each case. Note that occlusion could be easily incorporated into our approach, by allowing the input annotations to mark open as well as closed silhouette curves; we ignored this possibility for simplicity of description and implementation. Nevertheless our template model does contain extrusions for the pigeon legs, and these are satisfactorily modelled in many of the example images. For this pigeon dataset, we find a morphable model with 7 parameters in 3 hours. We use the weights $\sigma_{\text{norm}} = 5$, $\xi_0 = 0.25$ and $\xi_{\text{def}} = 0.05$. Example reconstructions are shown in Figure 6 and the accompanying video.

We also built a dataset from images of 20 polar bears, to test our framework on an animal with much more complicated pose variation. Here our method is limited by the lack of representation for articulated joints, and several reconstructions spill over the edge of the silhouette: an error which is not explicitly penalized by our reconstruction energy E. As the optimizer finds it very difficult to make progress on this dataset, we are able to build a 10-parameter model in 3 hours, using weights $\sigma_{\text{norm}} = 4$, $\xi_0 = 0.25$ and $\xi_{\text{def}} = 0.25$. Figure 14 shows a few reconstructions from the model.

Figure 7 shows some of the incremental improvements made by different parts of this paper. The results shown in (b), (c) and (d) were generated in 9, 21 and 24 hours, respectively. This difference between (b) and (c) is not explained directly by the running time of our global search, however, as the computation for this stage is negligible compared to the continuous optimization (see §5.2). The impact on the running time of the algorithm is therefore better explained by the fact that the global search, in (c), provides the continuous optimization with the opportunity to make a greater number of productive steps towards a minimum.

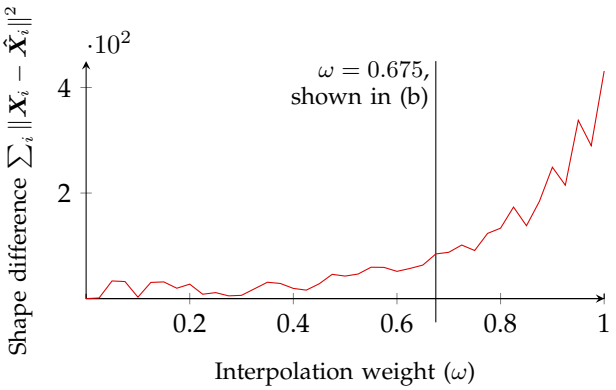
6.1 Sensitivity to template mesh

We investigated the sensitivity of the optimization to the initial template using two tests: first, dependence on mesh *geometry*, then mesh *topology*.

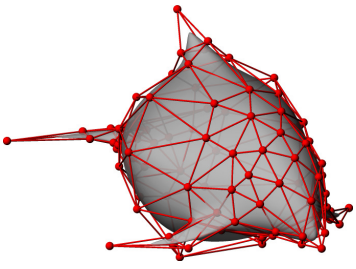
6.1.1 Geometry

To assess dependence on mesh geometry, we deformed the initial user-created estimate shown in Figure 2 to a sphere, by simply projecting the control vertex limit positions onto a sphere. A linear combination of the reference and sphere meshes, parametrized by a scalar ω, allows us to smoothly vary the deviation from the user-created initial estimate.

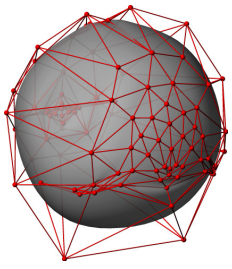
We then run our optimization for varying values of ω and record the 3D difference between the converged mesh and that reached for $\omega = 0$. The results are plotted in Figure 9a and show that we can start the optimization from a wide range of shapes, up to the blowfish-like geometry shown in Figure 9b, and find a similar result at the end of the optimization. The result worsens if we start from a very spherical shape



(a) Variation of model quality with initial estimate.

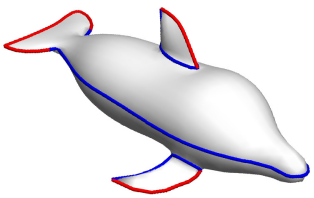


(b) Initialization for $\boldsymbol{B}_0$ when $\omega = 0.675$

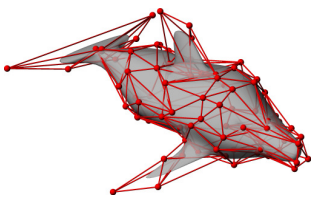


(c) Initialization for $\boldsymbol{B}_0$ when $\omega = 1$

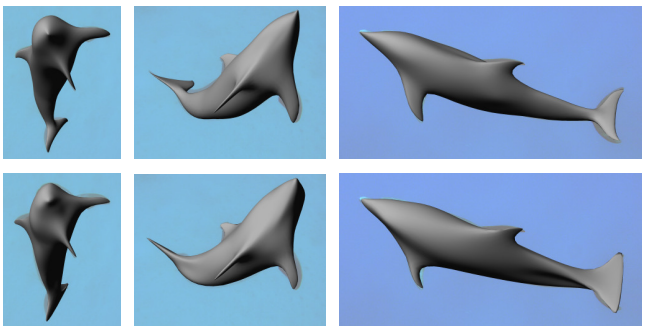
Fig. 9. Model quality plotted as a function of how sphere-like the initialization template is set. In (a) we compare the final shapes $\{\boldsymbol{X}_i\}_{i=1}^n$ against the shapes $\{\hat{\boldsymbol{X}}_i\}_{i=1}^n$ that are given by the result when $\omega = 0$. The final result is similar for a large range of values for ω, including the value $\omega = 0.675$ shown in (b).



(a) Model created in FiberMesh by sketching the blue (smooth) and red (sharp) contours.



(b) Shape shown in (a), decimated and converted to a Loop subdivision surface (c.f. Fig. 2).



(c) Comparison of reconstructions using (top row) the hand-crafted template mesh shown in Figure 2, and (bottom row) the template mesh shown in (b), created using FiberMesh.

Fig. 10. The single template mesh can be created using very little user interaction. Here we show the result of using FiberMesh [19] to create a template mesh, with which we reconstruct the same 32 dolphins used to create Figure 1. The reconstruction comparison in (c) shows that the resulting model is of a quality which is similar, but not completely equivalent, to the model built using the hand-crafted template shown in Figure 2.

(ω close to 1, as shown in Figure 9c), because the surface normals no longer provide useful information to the global search for contour generators. However, these results show that the method is quite tolerant of variations in starting geometry. Note that to give tractable computation times, this experiment was run with lowered optimization tolerances, just 16 images, and five basis shapes ($D = 4$).

6.1.2 *Topology*

A potentially more serious dependence is on the topology of the initial mesh. The mesh shown in Figure 2 contains some vertices which are clearly placed in order to capture some key dolphin features. While such meshes are easy to build for 3D modellers, we would like to test whether a more naive initial mesh can be used. We used FiberMesh [19] to build an approximate dolphin model, and then used a simple decimation strategy to define the subdivision surface control mesh, illustrated in Figure 10b. This mesh is of a rather lower quality than the hand-crafted template: triangles have more uneven aspect ratios, and there are a greater number of surface singularities. It would be possible to approximate the shape shown in Figure 10a without these problems, by using a more specialized subdivision-surface fitter [38]. Nevertheless, the model is not too badly affected by the poor quality of the starting mesh. The reconstructions shown in Figure 10c are typical in showing that the model built using a sketched template is close to the one shown in Figure 1, but suffers around the tail region, for example, where the sketched template provides fewer degrees of freedom (i.e. control vertices) to match the image data.

6.2 Sensitivity to weights on thin-plate energy

The weights ξ_0 and ξ_{def} determine the smoothness of the desired object class and deformations. These are user-determined parameters, so we need to ensure that their setting is a straightforward process. In Figure 11 we show the effect of changing these weights on three different object classes, demonstrating that the model changes smoothly in response. This figure also shows the importance of the terms E^{tp} to gain a good reconstruction; in particular, a value for ξ_0 which is too low leads to a highly implausible model, satisfying the constraints along each contour generator but with extreme, non-smooth changes in between.

6.3 Sensitivity to initial rotation

To test sensitivity to the user-provided view shown in Figure 4a, we added a random perturbation to each initial rotation $\hat{\boldsymbol{R}}_i$ for 16 dolphin instances, and found $\{\lambda_i, \boldsymbol{t}_i\}$ by solving a least-squares problem for the point constraints. Figure 12 shows that the number of point constraints in this dataset make the system very

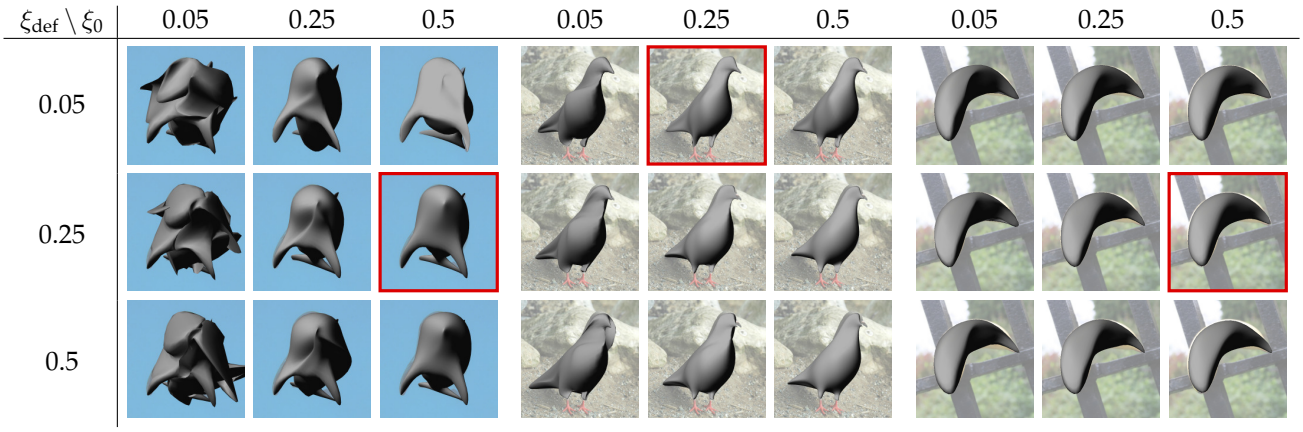


Fig. 11. Sensitivity to changes in weights on E^{tp}. This figure shows an example reconstruction from models built using 32 dolphins, 25 pigeons and 10 bananas, with a variety of values for ξ_0 and ξ_{def}. Note that the leftmost column shows a very low value of ξ_0, illustrating the need for the smoothness terms. For each dataset we highlight in red the model that is used in the rest of the paper.

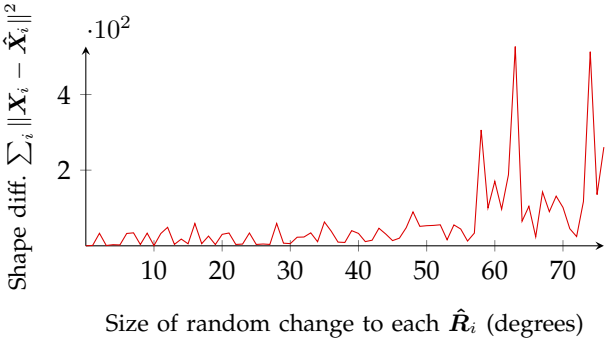


Fig. 12. Variation of model quality with perturbed initial rotations $\{\hat{\boldsymbol{R}}_i\}_{i=1}^n$. This graph plots the difference between the final shapes $\{\boldsymbol{X}_i\}_{i=1}^n$ and the shapes $\{\hat{\boldsymbol{X}}_i\}_{i=1}^n$ that are given by the unperturbed rotations.

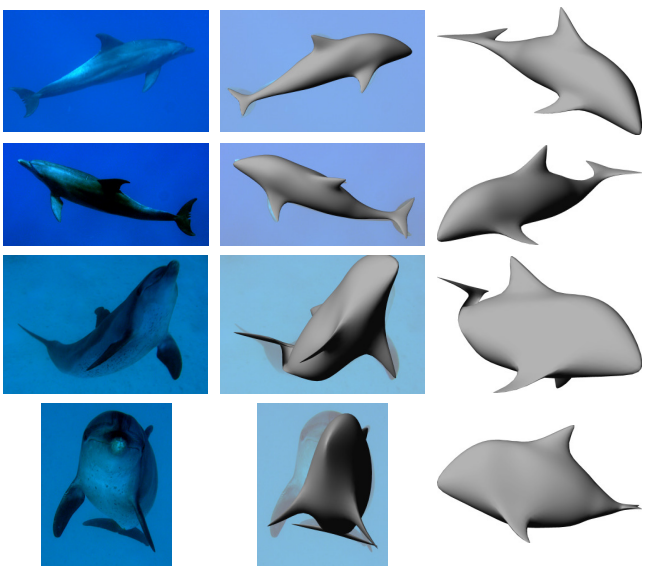


Fig. 13. Reconstructions that solve for each image separately, by using fixed eigenmodes of E^{tp} as the basis shapes $\boldsymbol{B}_m$. In each row we show (left) the original image, (middle) the eigenmode reconstruction for 5 modes in each of three coordinate directions, i.e. $D = 15$ and (right) another view of the same reconstruction. While some configurations (e.g. the top two images) can give reasonable results for a single image and fixed basis shapes, others (e.g. the bottom two) need the information available from other images to obtain plausible reconstructions.

insensitive to changes in the supplied rotations; even random perturbations of 50 degrees make little change to the final reconstructed shapes. This experiment used the same settings described in §6.1.1 to generate models with five basis shapes.

6.4 Comparison to fixed deformation modes

In this work we use the information available from multiple images of an object class to constrain the model (1). However, the discrete search described in §5.2 decouples the separate problems to be solved in each image, and we could do the same for the continuous optimization described in §5.1, if it were possible to fix the basis shapes $\boldsymbol{B}_m$. It is interesting to consider if similar results can be obtained by solving these much smaller optimization problems.

In Figure 13, we show results of an experiment which uses eigenmodes of the smoothness energy E^{tp} as the basis shapes $\boldsymbol{B}_m$. Since the thin-plate energy is a sum of independent terms in each coordinate direction, the eigenmodes are independent and we assign basis shapes for deformation in each coordinate direction. We modify E^{reg} to penalize these modes by weights equal to the corresponding eigenvalues, so the resulting model is exactly a low-order eigenbasis approximation to the thin plate smoothness energy. While it would be possible to improve these results by providing a shape basis which gives more plausible deformations, we use the same thin plate smoothness measure in the rest of the paper. Any deficiencies in the smoothness model are therefore shared by all of our results, and yet the reconstructions shown in Figure 13 are far less plausible than those shown in Figure 1, for example. In some images the silhouette constrains the model reasonably well, and so we can find quite good single-view reconstructions (see top two rows of Figure 13). However, other images are far more reliant on the shape constraints imposed by multiple silhouettes, and so the resulting single-view

reconstructions are highly implausible (see bottom two rows of the same figure).

7 Conclusions

We have shown that models can be constructed for non-rigid object classes purely from 2D images, given an initialization using a coarse rigid model. The user input required is quite simple, and might even be considered comparable to the effort required to construct a morphable model using range images, if one considers the effort of positioning the scanner and generating mesh correspondences. Because our correspondences are an intrinsic part of the optimization, guided by a few point constraints, we recover the morphable model directly, rather than by first computing 3D and then finding 3D correspondences. This is almost certainly the first system to recover nonrigid shape from silhouettes in multiple unorganized views, and even its restriction to rigid geometry is a novel contribution. In addition we have contributed innovations in the use of subdivision surfaces in fitting to image data. Many further questions naturally arise from this work, some of which we address here.

The first is: what makes an object class? One might imagine building class models of 'bottlenose dolphins', 'sea mammals', or even 'fish and the like'. How are we to choose the granularity? First, a larger, more complex class will require more images to build a comprehensive model, so limitations on the number of available images and computation power might argue for a smaller class. Also, a large class may not fit the unimodal Gaussian distribution inherently assumed by the linear morphable model, so to model larger classes we might require a more complex generating distribution, such as a Gaussian process. However, there are benefits to considering larger sets. For example, a hierarchical approach, where basis shapes are shared between different species, and then subspecies, might overcome a lack of data for rare species. For example, if one has hundreds of images of all sorts of fish, whales, and dolphins, an accurate bottlenose dolphin model might be obtained from a few extra images by adding a small number of specialized basis shapes to the generic model.

Another natural question is whether the requirement for a coarse initialization can be relaxed. In principle the system should be able to converge from a sphere, but a key extension that would be required is for the optimizer to be able to change the control mesh triangulation, which is not currently implemented. It may be that another discrete optimization stage would permit this adaptation, generalizing the system. On the other hand, creating one approximate rigid model is considerably simpler than building the entire model. With systems such as FiberMesh, the former is quite easy, while the latter is difficult even for expert artists. Additionally, as mentioned in §6, we could enrich the range of annotations we allow, perhaps allowing the user to mark contour generator discontinuities or crease edges, or intra-class texture edges like Prasad et al. [24]. In each case, these annotations are easy for a human to provide [15], being essentially symbolic rather than numeric, but would increase the scope and robustness of the system. Another possible future application is to video sequences rather than unrelated images. Like Furukawa et al. [8], in this context it should be possible to use coherence between consecutive frames to give better reconstructions.

Fig. 14. Images showing three polar bears out of 20, from which we build a 10-parameter morphable model. Top: input images with silhouette annotations. Bottom: final morphable model reconstruction overlaid on the input images.

Several limitations of the current approach could be addressed. The technique is not yet well suited to the recovery of articulated models, as exemplified by the polar bears in Figure 14. An interesting direction might be to look at the analogues of the deformable case which exist for range data [39], or to consider an extension to parametrized shapes such as chairs and road vehicles. Another failure mode is 'spillage', where the surface obeys the silhouette incidence and normal constraints at the contour generator, but another part of the surface falls outside the imaged object. It may be possible to avoid this by augmenting the energy with an image-based penalty term or by using constrained optimization, gaining better models but with the risk of introducing new local optima.

Acknowledgements

This work was partially supported by the SNF under project number 200021-134639.

We thank the following copyright holders for permission to reproduce their photographs in this paper: Tim Nicholson for all dolphins except leftmost and rightmost images in Figure 1; Peter Asprey[1] for left-

1. Image copyright Peter Asprey, Derby, UK, July 14 2007.

most image in Figure 1; Arnaud Clerget for rightmost image in Figure 1; Helen White for all photographs of pigeons; Colin Burnett for Figure 14 (left)[2]; Trisha Shears for Figure 14 (middle); Pcb21 for Figure 14 (right)[2].

REFERENCES

[1] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3D faces," in *Proc. SIGGRAPH*, A. Rockwood, Ed., 1999, pp. 187–194.

[2] B. Allen, B. Curless, and Z. Popović, "The space of human body shapes: reconstruction and parameterization from range scans," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 587–594, 2003.

[3] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "SCAPE: Shape Completion and Animation of People," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 408–416, 2005.

[4] A. Balan and M. J. Black, "The naked truth: Estimating body shape under clothing," in *Proc. European Conf. on Computer Vision, Part II*, D. Forsyth, P. Torr, and A. Zisserman, Eds., 2008, pp. 15–29.

[5] S. Zhou, H. Fu, L. Liu, D. Cohen-Or, and X. Han, "Parametric reshaping of human bodies in images," *ACM Trans. Graph.*, vol. 29, no. 3, pp. #126:1–10, 2010.

[6] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3D," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 835–846, 2006.

[7] L. Torresani, A. Hertzmann, and C. Bregler, "Non-rigid structure-from-motion: Estimating shape and motion with hierarchical priors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 878–892, 2008.

[8] Y. Furukawa, A. Sethi, J. Ponce, and D. J. Kriegman, "Robust structure and motion from outlines of smooth curved surfaces," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 2, pp. 302–315, 2006.

[9] P. McIlroy and T. Drummond, "Reconstruction from uncalibrated affine silhouettes," in *Proc. British Machine Vision Conference*, 2009, pp. 1–11.

[10] M. Prasad, A. Zisserman, and A. Fitzgibbon, "Single view reconstruction of curved surfaces," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2006, pp. 1345–1354.

[11] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, "Active shape models—their training and application," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 38–59, 1995.

[12] T. Vetter and T. Poggio, "Linear object classes and image synthesis from a single example image," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 733–742, 1997.

[13] Y. Chen, T. Kim, and R. Cipolla, "Inferring 3D shapes and deformations from single views," in *Proc. European Conf. on Computer Vision, Part III*, K. Daniilidis, P. Maragos, and N. Paragios, Eds., 2010, pp. 300–313.

[14] D. Terzopoulos, A. Witkin, and M. Kass, "Symmetry-seeking models and 3D object reconstruction," *International Journal of Computer Vision*, vol. 1, no. 3, pp. 211–221, 1988.

[15] Y. Gingold, T. Igarashi, and D. Zorin, "Structured annotations for 2D-to-3D modeling," *ACM Trans. Graph.*, vol. 28, no. 5, pp. #148:1–9, 2009.

[16] T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A sketching interface for 3D freeform design," in *Proc. SIGGRAPH*, A. Rockwood, Ed., 1999, pp. 409–416.

[17] O. A. Karpenko and J. F. Hughes, "SmoothSketch: 3D free-form shapes from complex sketches," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 589–598, 2006.

[18] X. Gu, S. J. Gortler, and H. Hoppe, "Geometry images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 355–361, 2002.

[19] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "FiberMesh: Designing freeform surfaces with 3D curves," *ACM Trans. Graph.*, vol. 26, no. 3, pp. #41:1–10, 2007.

[20] V. Kraevoy, A. Sheffer, and M. van de Panne, "Modeling from contour drawings," in *Proceedings of the Eurographics Symposium on Sketch-Based Interfaces and Modeling*. ACM, 2009, pp. 37–44.

[21] J. Porrill and S. Pollard, "Curve matching and stereo calibration," *Image and Vision Computing*, vol. 9, no. 1, pp. 45–50, 1991.

[22] R. Cipolla, K. E. Åström, and P. J. Giblin, "Motion from the frontier of curved surfaces," in *Proc. Fifth International Conference on Computer Vision*. IEEE, 1995, pp. 269–275.

[23] C. Bregler, A. Hertzmann, and H. Biermann, "Recovering non-rigid 3D shape from image streams," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2000, pp. 690–696.

[24] M. Prasad, A. W. Fitzgibbon, A. Zisserman, and L. Van Gool, "Finding Nemo: Deformable object class modelling using curve matching," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[25] S. Ilić, M. Salzmann, and P. Fua, "Implicit meshes for effective silhouette handling," *International Journal of Computer Vision*, vol. 72, no. 2, pp. 159–178, 2007.

[26] L. Kobbelt, S. Campagna, and H.-P. Seidel, "A general framework for mesh decimation," in *Graphics Interface*, 1998, pp. 43–50.

[27] M. Prasad, A. Zisserman, A. W. Fitzgibbon, M. P. Kumar, and P. H. S. Torr, "Learning class-specific edges for object detection and segmentation," in *Proc. Indian Conf. on Computer Vision, Graphics and Image Processing*, 2006.

[28] D. Dementhon and L. Davis, "Model-based object pose in 25 lines of code," *International Journal of Computer Vision*, vol. 15, no. 1–2, pp. 123–141, 1995.

[29] E. Tosun, "Geometric modeling using high-order derivatives," Ph.D. dissertation, New York University, 2008.

[30] M. Halstead, M. Kass, and T. DeRose, "Efficient, fair interpolation using Catmull-Clark surfaces," in *Proc. SIGGRAPH*, J. T. Kajiya, Ed., 1993, pp. 35–44.

[31] F. Grassia, "Practical parameterization of rotations using the exponential map," *Journal of Graphics Tools*, vol. 3, no. 3, pp. 29–48, 1998.

[32] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment — a modern synthesis," in *Vision Algorithms: Theory and Practice*, ser. LNCS, B. Triggs, A. Zisserman, and R. Szeliski, Eds., 2000, vol. 1883, pp. 298–372.

[33] B. Appleton and C. Sun, "Circular shortest paths by branch and bound," *Pattern Recognition*, vol. 36, no. 11, pp. 2513–2520, 2003.

[34] C. T. Loop, "Smooth subdivision surfaces based on triangles," *Master's thesis, University of Utah*, Aug 1987.

[35] J. Stam, "Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values," in *Proc. SIGGRAPH*, M. Cohen, Ed. ACM, 1998, pp. 395–404.

[36] J. Peters and U. Reif, *Subdivision Surfaces*. Springer, 2008.

[37] D. Zorin, "Constructing curvature-continuous surfaces by blending," in *Proceedings of Symposium on Geometry Processing*, K. Polthier and A. Sheffer, Eds. Eurographics, 2006, pp. 31–40.

[38] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, "Piecewise smooth surface reconstruction," in *Proc. SIGGRAPH*, A. Glassner, Ed., 1994, pp. 295–302.

[39] D. Anguelov, D. Koller, H.-C. Pang, P. Srinivasan, and S. Thrun, "Recovering articulated object models from 3D range data," in *Proc. Uncertainty in AI*, 2004, pp. 18–26.

2. Used under a Creative Commons BY-SA 3.0 License. *http://creativecommons.org/licenses/by-sa/3.0/*

Thomas Cashman is a postdoctoral researcher at the University of Lugano. He completed his PhD at the University of Cambridge in 2010. His thesis received the Eurographics PhD Award and was runner-up in the BCS Distinguished Dissertation competition.

Andrew Fitzgibbon is a principal researcher at Microsoft Research, Cambridge, UK, where he works on new ways of thinking about video, particularly video of moving objects such as people, animals, trees and water. He has received several awards for his academic papers, including twice receiving the IEEE's Marr Prize; and software he wrote won an Engineering Emmy Award in 2002 for significant contributions to the creation of complex visual effects.

What Shape are Dolphins? Building 3D Morphable Models from 2D Images

Supplementary material

Thomas J. Cashman and Andrew W. Fitzgibbon, *Senior Member, IEEE*

Abstract—Our paper [1] shows that there is enough information in a collection of 2D pictures of certain object classes to generate a full 3D morphable model, even in the absence of surface texture. This supplementary material provides technical details on the system we implemented.

The morphable models that we build are a linear combination of subdivision surfaces, and an important component of the optimization problem is finding each object's contour generators in the surface parameter space. To do so, this document introduces two novel constructions on subdivision surfaces: a local distance measure and a local reparametrization around extraordinary vertices.

We also describe the structure of the Jacobian that appears in the continuous optimization subproblems that we solve, and in an appendix we give the function which calculates the product of that Jacobian with another matrix without forming the Jacobian matrix explicitly.

1 Introduction

This supplementary material accompanies our paper 'What Shape are Dolphins? Building 3D Morphable Models from 2D Images' [1]. For a glossary of notation used in that paper and this document, see Table 1.

2 Parameter-based distance measure

In order to evaluate E^{cg}, introduced in §4.3 of our paper [1], we need a function $d(\mathring{u}_j, \mathring{u}_{j+1})$ which serves as a distance measure between positions in Ω. The ideal definition, to be independent of the surface parametrization, would be the geodesic distance between $M(\mathring{u}_j)$ and $M(\mathring{u}_{j+1})$. However d is evaluated in the innermost loop of our optimization, and geodesic distances for freeform surfaces are expensive to compute [2]. We therefore settle for a purely parameter-based measure, independent of the control vertices $\boldsymbol{X}$, which will be a good approximation to geodesic distance if the control mesh is sufficiently uniform and planar. By depending solely on the mesh topology we can evaluate d just once, and cache the resulting values, for each pair of candidate points in Φ.

To build this parameter-based distance measure, we need a way to handle the structure of Ω, which is not a single global space but consists of multiple pieces. Fortunately, since the output of d is truncated by τ (see §4.3 in our paper [1]), we only need d defined where a pair of points is sufficiently close in Ω. This allows us to avoid difficult questions surrounding the global surface topology. There is also the effect of the mesh connectivity to consider, since M is always a non-isometric map near *extraordinary vertices*: those with edges connecting to fewer or greater than six other vertices in the control mesh. The number of edge-connected vertices is the *valency* of a vertex.

Here we use a similar technique to Zorin [3], associating a region surrounding each vertex of valency r with the associated characteristic map $\boldsymbol{\psi}_r$ [4]. The characteristic map is defined from the subdominant eigenvectors of the matrix representation of a subdivision step; see Figure 3a for an example. It is relevant here because an affine transformation of $\boldsymbol{\psi}_r$ gives a first-order Taylor approximation of a subdivision surface at every singularity with valency r. By measuring distance in the image of parameter space under this characteristic map, we can compensate for the scaling of the surface, with respect to the parametrization, which always occurs around extraordinary vertices.

In each triangle surrounding a vertex v of the control mesh, let $\boldsymbol{A}(v)$ be the parametric Voronoi cell associated with v, i.e. the region where the barycentric coordinate associated with v is greater than those associated with the other two vertices in the triangle (see Figure 1). Then we evaluate $d(\mathring{u}_j, \mathring{u}_{j+1})$ using the following algorithm, where the cases correspond to Figure 1:

1) If $\mathring{u}_j$ and $\mathring{u}_{j+1}$ belong to a common cell $\boldsymbol{A}(v)$ and r is the valency of v, let $d(\mathring{u}_j, \mathring{u}_{j+1}) = \|\boldsymbol{\psi}_r(\mathring{u}_j) - \boldsymbol{\psi}_r(\mathring{u}_{j+1})\|$.

- *T. Cashman is with the Faculty of Informatics, University of Lugano, Switzerland. E-mail: thomas.cashman@usi.ch.*
- *A. Fitzgibbon is with Microsoft Research, 7 JJ Thomson Ave, Cambridge, UK. E-mail: awf@microsoft.com.*

Indices and constants

$h \in \mathbb{R}^+$	Threshold for d used by τ: see §4.3
$i \in 1 \,..\, n$	Indexes images: see §3
$j \in 1 \,..\, S_i$	Indexes silhouette samples in image i
$k \in 1 \,..\, K_i$	Indexes point constraints in image i
$m \in 0 \,..\, D$	Indexes basis shapes: see (1)
$p \in \mathbb{N}$	Number of vertices in subdivision surface control mesh: see §2
$S = \sum_i S_i$	Total number of silhouette samples
$\sigma_{\text{sil}} \in \mathbb{R}$	Estimate of noise in silhouettes: see (5)
$\sigma_{\text{con}} \in \mathbb{R}$	Estimate of noise in point constraints: see (6)
$\sigma_{\text{norm}} \in \mathbb{R}$	Estimate of noise in normals: see (7)
$\beta \in \mathbb{R}^+$	Constant that fixes the scale of the deformation basis shapes: see (9)
$\gamma \in \mathbb{R}^+$	Weight on terms that enforce piecewise contour generator continuity: see (10)
$\xi_0 \in \mathbb{R}$	Optimization weight on smoothness of target object class: see §4.4
$\xi_{\text{def}} \in \mathbb{R}$	Optimization weight on smoothness of model deformations: see §4.4

Problem specification

$\boldsymbol{s}_{ij} \in \mathbb{R}^2$	Silhouette sample j in image i: see §3.1
$\boldsymbol{s}_i \in \mathbb{R}^{2S_i}$	Silhouette in image i: $\boldsymbol{s}_i = \{\boldsymbol{s}_{ij}\}_{j=1}^{S_i}$
$\boldsymbol{n}_{ij} \in \mathbb{R}^2$	Normal of silhouette sample j in image i: see §3.1
$\mathring{\mu}_{ik} \in \Omega$	Preimage of point constraint k in image i: see (2)
$\boldsymbol{c}_{ik} \in \mathbb{R}^2$	The kth point constraint in image i: see (2)

Shape and pose parameters P

$\alpha_{im} \in \mathbb{R}$	Coefficient of basis shape m in image i: see (1)
$\boldsymbol{B}_m \in \mathbb{R}^{3p}$	Control vertices for mth basis shape: see (1)
$\boldsymbol{R}_i \in \mathbb{R}^{3\times3}$	Rotation for camera viewing image i: see (3)
$\lambda_i \in \mathbb{R}$	Scale for camera viewing image i: see (3)
$\boldsymbol{t}_i \in \mathbb{R}^2$	Translation for camera viewing image i: see (3)
$\boldsymbol{\theta}_i \in \mathbb{R}^3$	Vector used to optimize $\boldsymbol{R}_i$: see §5.1
$\boldsymbol{X}_i \in \mathbb{R}^{3p}$	Control vertices for instance in image i: see (1)

Preimage parameters U

$\mathring{u}_{ij} \in \Omega$	Contour preimage sample j in image i: see (4)
$\mathring{\boldsymbol{u}}_i \in \Omega^{S_i}$	Contour preimage in image i: $\mathring{\boldsymbol{u}}_i = \{\mathring{u}_{ij}\}_{j=1}^{S_i}$

Functions and parametric domains

$\Omega = \triangle \times [\,1 \,..\, g\,]$	Subdivision surface parameter space: there is a unit triangle $\triangle$ for each of g control mesh triangles (see Figure 2)
$M : \Omega \to \mathbb{R}^3$	Given control vertices $\in \mathbb{R}^{3p}$, evaluates subdivision surface: see §2
$N : \Omega \to \mathbb{R}^3$	Given control vertices $\in \mathbb{R}^{3p}$, evaluates subdivision surface normal: see §2
$\Phi \subset \Omega$	Discrete candidate positions for contour preimage samples $\mathring{u}_{ij}$ (see §5.2)
$\boldsymbol{\psi}_r : \triangle \times [\,1 \,..\, r\,] \to \mathbb{R}^2$	Characteristic map for subdivision surface singularity with valency r
$d : \Omega \times \Omega \to \mathbb{R}$	Parametric distance measure: see §4.3
$\tau : \mathbb{R} \to \mathbb{R}$	Quadratic truncated at h^2: see §4.3
$\pi_i : \mathbb{R}^3 \to \mathbb{R}^2$	Camera projection for image i: see (3)

TABLE 1. Glossary of notation. All references are to our paper [1].

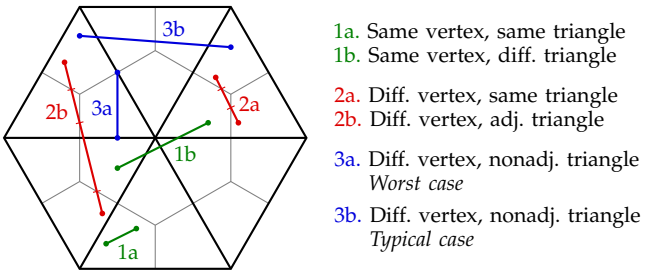


Fig. 1. Example configurations encountered during the computation of d. The Voronoi cells for each vertex are marked with light gray lines. Intersections with these cells are marked where they are calculated by our algorithm. For the line marked 3a, the points are positioned as close as possible while falling in case 3. In this configuration, the distance computed by d is greater than $\sqrt{3}/4$ as long as the valency of the adjacent vertex is six or greater.

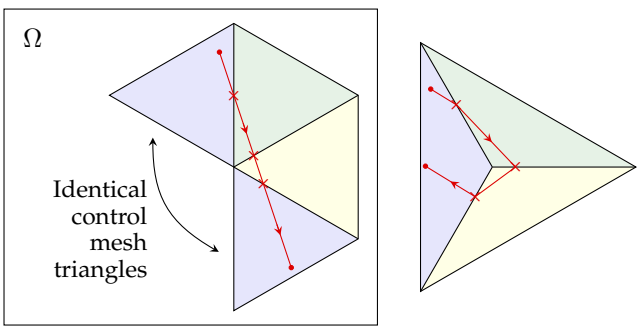


Fig. 2. Without reparametrization, parameter space updates are distorted by extraordinary vertices. In this example, a valency 3 vertex distorts an update which is a straight line in parameter space (left) into a complete U-turn around the valency 3 vertex in the control mesh (right). Each intersection with a parameter space triangle, leading to a transition to an adjacent triangle, is marked ×. We avoid this distortion by reparametrizing near extraordinary vertices.

2) Otherwise, let T_j be the triangle in which $\mathring{u}_j$ falls. If T_j and T_{j+1} are the same triangle, or adjacent triangles joined by an edge, then consider the planar embedding of the equilateral parametric triangles T_j and T_{j+1}. Partition the line from $\mathring{u}_j$ to $\mathring{u}_{j+1}$ into (at most four) segments, each of which lies in a single cell $\boldsymbol{A}(w)$ for some vertex w. Let $d(\mathring{u}_j, \mathring{u}_{j+1})$ be the sum of d calculated on each of these line segments using case 1 above.
3) Otherwise (if $\mathring{u}_j$ and $\mathring{u}_{j+1}$ do not lie in a common cell $\boldsymbol{A}(v)$, and also do not lie in the same or adjacent triangles), then $d(\mathring{u}_j, \mathring{u}_{j+1}) = \infty$. That is, these points are considered too far away from each other on the surface, and so there is a fixed penalty h^2 in E^{cg}.

This last case serves two purposes. It ensures that d is inexpensive to evaluate, as there is no need to search for an optimal path between $\mathring{u}_j$ and $\mathring{u}_{j+1}$. It also guarantees that d is well-defined and symmetric: a longer path between distant points on the surface may not even have an unfolding into the plane. Furthermore, by setting h (the threshold value of τ) to no larger than $\sqrt{3}/4$, we can make sure that the topological limitation imposed in case 3 almost never creates a discontinuity in E^{cg} (see Figure 1). If a larger threshold is required for an image i, then an equivalent effect can be achieved by increasing S_i (i.e. by taking a denser set of silhouette samples).

3 Updates to contour preimage

Using a subdivision surface also makes it challenging to apply updates to the contour preimage points $\mathring{u}_{ij}$, since update vectors computed by the optimizer may require moving between the separate triangles that make up the parameter space. At each iteration of the continuous optimization described in §5.1 of our paper [1], let $\delta\mathring{u}_{ij} \in \mathbb{R}^2$ be the update vector that is applied to contour preimage sample $\mathring{u}_{ij}$. We also define $\tilde{u}_{ij} \in \triangle$ as the barycentric coordinate part of $\mathring{u}_{ij}$, giving 2D coordinates in the unit triangle $\triangle$. This parameter space triangle corresponds to the triangle in the control mesh with index T_{ij}.

We know that we need to modify T_{ij} (i.e. transition to a different triangle in the control mesh) if $\tilde{u}_{ij} + \delta\mathring{u}_{ij} \notin \triangle$. Marinov and Kobbelt [5] address exactly this issue to compute closest-point queries. Their approach limits each optimizer iteration to updates within a parameter triangle, transitioning to an adjacent triangle only if a point already lies on the boundary. That is, if $\tilde{u}_{ij} + \delta\mathring{u}_{ij} \notin \triangle$, they compute an intersection with the boundary $\partial\triangle$ of the unit triangle, by finding a real value $0 \leq t < 1$ such that $\tilde{u}_{ij} + t\delta\mathring{u}_{ij} \in \partial\triangle$. $\mathring{u}_{ij}$ is then updated with the shortened vector $t\delta\mathring{u}_{ij}$ instead. They modify T_{ij} only if $\tilde{u}_{ij} \in \partial\triangle$ and $\tilde{u}_{ij} + \delta\mathring{u}_{ij} \notin \triangle$. Using this approach, an update from the interior of one triangle to the interior of another requires 3 optimizer iterations, and moving across a vertex of valency r requires at least $r + 1$.

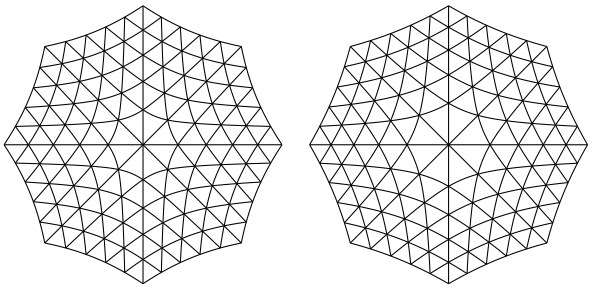
For a single parameter point this is simple and effective, but in our case the optimizer is modifying all available variables simultaneously. Therefore

- iterations are computationally expensive: it is important to use as few of them as possible;
- delaying the convergence of a sample $\mathring{u}_{ij}$ will have an effect on every other variable: unlike closest-point queries, an incomplete iteration is not free of side effects.

We therefore want to allow parameter-space updates to converge in as few iterations as possible. To do so, we can compute an intersection with $\partial\triangle$, as before, but then modify T_{ij} and interpret the remaining part of the update vector, $(1-t)\delta\mathring{u}_{ij}$, as an update to be computed in the adjacent triangle. We can continue in this way, possibly computing more intersections with triangle boundaries as necessary, until the entire update has been applied. This still ensures that the new value for $\tilde{u}_{ij} \in \triangle$, and this approach places no restrictions on the new value for T_{ij}, other than the fact that there must be a straight path, in parameter space, from the old position of $\mathring{u}_{ij}$ to the new.

Extraordinary points present a problem for this approach, however, as they distort nearby parameter updates. Using this interpretation of a 'straight path in parameter space' near a valency 3 vertex, for example, can lead to large updates which leave and then return to the same triangle, or even move the contour preimage sample in the opposite direction (see Figure 2). A solution to this distortion comes from applying the same idea as in §2: we reparametrize the region surrounding each extraordinary vertex. To use this approach, we define a circular region $\bigcirc$ at a fixed parametric radius around each extraordinary vertex. The radius we use is half the length of a parametric edge. Now for each parameter-space update, we calculate the intersection with the reparametrization boundary $\partial\bigcirc$ as well as with the triangle boundary $\partial\triangle$. If the first intersection is with $\partial\triangle$, then the transition to an adjacent triangle is handled as before. Otherwise, we update $\mathring{u}_{ij}$ so that $\tilde{u}_{ij} \in \partial\bigcirc$, and calculate $\hat{u}_{ij} = \boldsymbol{\psi}_r(\tilde{u}_{ij})$, where r is the valency of the adjacent extraordinary vertex. Now $(1-t)\delta\mathring{u}_{ij}$ is treated as an update in the *reparametrized region* $\boldsymbol{\psi}_r(\bigcirc)$ instead. To find the final position on the surface, we calculate $\tilde{u}_{ij} = \boldsymbol{\psi}_r^{-1}(\hat{u}_{ij})$, where $\hat{u}_{ij}$ is the final position of the sample in the reparametrized coordinates.

This allows a seamless transition from one side of an arbitrary-valency vertex to the other. Unfortunately $\boldsymbol{\psi}_r^{-1}$, the inverse of the characteristic map, is complicated and expensive to compute [6]. Instead, we approximate the characteristic map using the *fractional power embedding* [7]: the symmetric extension of the map $z \mapsto z^{(6/r)}$ when evaluated on the first $60°$ sector of the complex plane. The inverse of this map is



(a) ψ_8 (b) $z \to z^{(6/8)}$

Fig. 3. A fractional power embedding such as (b) is a reasonable approximation to the corresponding characteristic map (a), but the inverse of (b) is straightforward to compute.

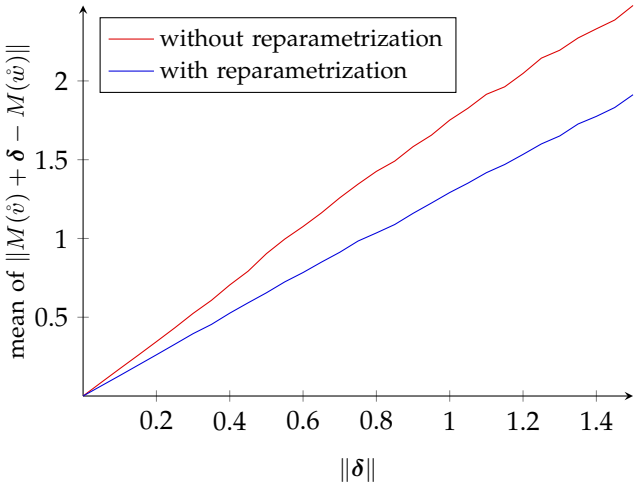
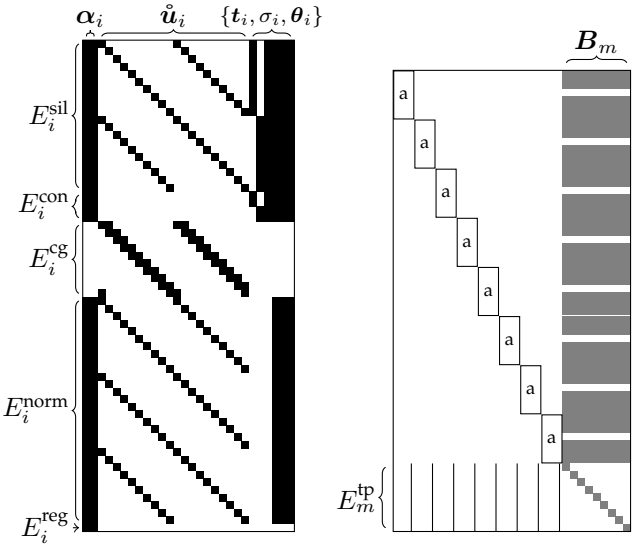


Fig. 4. We improve parameter updates using local reparametrizations surrounding extraordinary vertices. Here we show the improvement on the initial dolphin model shown in Figure 2, by taking a random point $\mathring{v} \in \Omega$ and a random update vector $\boldsymbol{\delta}$ which lies in the tangent plane of $M(\mathring{v})$. We apply the parameter update, with and without reparametrization, to gain a new position on the surface $M(\mathring{w})$. This plot shows the mean distance between the target point in the tangent plane, $M(\mathring{v}) + \boldsymbol{\delta}$, and the new point on the surface, $M(\mathring{w})$, for 10,000 such trials at each value for $\|\boldsymbol{\delta}\|$.

straightforward to compute, and it has the same r-fold rotational symmetry as the characteristic map (see Figure 3).

Each update of a parameter sample $\mathring{u}_{ij}$ attempts to move $M(\mathring{u}_{ij})$ towards a new target position, which lies in the tangent plane of the surface (when evaluated at $\mathring{u}_{ij}$), as the first derivatives are the only surface information available through the Jacobian. Figure 4 summarizes an experiment which demonstrates that our reparametrization gives a small but significant improvement in updating parameter points so that the resulting point on the surface is closer to the optimizer's target. This allows a better convergence of the contour preimage, and suggests that our technique could be useful wherever subdivision surfaces are applied in optimization.



(a) Block for a single image i. (b) Full Jacobian structure: gray regions indicate sparse blocks.

Fig. 5. Jacobian sparsity structure. For this simplified example $n = 8$, $D = 2$, $K_i = 2$ and $S_i = 10$ for all i. The structure of the complete Jacobian is shown schematically in (b), where the 8 blocks marked 'a' have the sparsity structure shown in (a).

4 Jacobian Structure

For continuous local optimization (as described in §5.1 of our paper [1]), we provide the optimizer with an analytic Jacobian, which grants a dramatic increase in performance over finite differencing. As the Jacobian matrix is sparse and structured (see Figure 5), we also save time and space by providing a function which calculates the product with another matrix without forming the Jacobian explicitly. This function appears as the Appendix of this document to show how we implemented the structure shown in Figure 5. The full MATLAB source code of our implementation is available from http://forms.codeplex.com/.

References

[1] T. J. Cashman and A. W. Fitzgibbon, "What Shape are Dolphins? Building 3D Morphable Models from 2D Images," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012.
[2] A. Spira and R. Kimmel, "An efficient solution to the eikonal equation on parametric manifolds," *Interfaces and Free Boundaries*, vol. 6, no. 3, pp. 315–327, 2004.
[3] D. Zorin, "Constructing curvature-continuous surfaces by blending," in *Proceedings of Symposium on Geometry Processing*, K. Polthier and A. Sheffer, Eds. Eurographics, 2006, pp. 31–40.
[4] U. Reif, "A unified approach to subdivision algorithms near extraordinary vertices," *Computer Aided Geometric Design*, vol. 12, no. 2, pp. 153–174, 1995.
[5] M. Marinov and L. Kobbelt, "Optimization techniques for approximation with subdivision surfaces," in *Proceedings of the Symposium on Solid Modeling and Applications*. Eurographics, 2004, pp. 113–122.
[6] I. Boier-Martin and D. Zorin, "Differentiable parameterization of Catmull-Clark subdivision surfaces," in *Proceedings of Symposium on Geometry Processing*. Eurographics, 2004, pp. 155–164.
[7] J. Peters and U. Reif, *Subdivision Surfaces*. Springer, 2008.

APPENDIX

```
function W = jacobmult(Jdata, X, flag)
    % vars is the complete set of optimization parameters (P and U in the paper)
    %     cM is the current number of parameters in the morphable model
    %     P is the number of vertices in the control mesh
    vars = Jdata(end, 1:2 * sum(T) + n * (7 + cM) + 3 * P * (cM + 1));

    % Derivatives are provided in the matrix 'Jdata' in a packed form. This
    % loop extracts the values ready for use in the functions below.
    meansc = 0;
    for i = 1:n
        % Start and end of derivative blocks for image i
        bs       = (i - 1) * (9 + cM + 3 * P) + 1;
        be       =       i * (9 + cM + 3 * P);

        % Global cell array holding derivatives with respect to the contour preimage
        %     S(i) is the number of silhouette samples
        %     T(i) is the number of free silhouette samples, taking into
        %         account any point constraints that lie on the silhouette
        silJ{i}  = Jdata(1:2 * S(i) + 5 * T(i), bs:bs + 1);

        % Global cell array holding derivatives with respect to the camera parameters
        %     and shape coefficients. K(i) is the number of point constraints for image i
        varsJ{i} = Jdata(1:    S(i) + 5 * T(i) + 2 * K(i), bs + 2:bs + 8 + cM);

        % Global cell array holding derivatives with respect to changes in the control mesh
        meshJ{i} = Jdata(1:    S(i) + 5 * T(i) + 2 * K(i), bs + 9 + cM:be);

        % Mean orthographic scale factor: sv(i) gives the index of the orthographic scale factor
        % for image i (lambda_i)
        meansc   = meansc + vars(sv(i));
    end
    meansc = meansc / n;

    % Global variables referenced in the functions below include:
    %      vs(i)           : parameters related to image i in vars have indices vs(i) + 1:vs(i + 1)
    %     Xvs(i), Xve(i)   : start and end indices for subranges of the above, for
    %                          X = s : contour preimage parameters
    %                          X = r : camera rotation parameters
    %                          X = m : shape mode coeffs (alpha)
    %
    %      es(i)           : equations related to image i in Jacobian have indices es(i) + 1:es(i + 1)
    %     Xes(i), Xee(i)   : start and end indices for subranges of the above, for
    %                          X = s : E^sil
    %                          X = n : E^norm
    %                          X = m : E^reg
    %                          X = c : E^cg
    %
    %     tplatesqrt_3     : sqrt(Q), as defined in Section 5.1 of the paper
    %     fixed_sil_pts{i} : boolean array that is true for silhouette samples that are given
    %                          by point constraints
    %     xi_0, xi_def     : weights on E^tp, as described in the paper

    switch sign(flag)
        case +1
            W =  jacobmultJX(X, vars, meansc);
        case -1
            W = jacobmultJtY(X, vars, meansc);
        case 0
            Y =  jacobmultJX(X, vars, meansc);
            W = jacobmultJtY(Y, vars, meansc);
    end

    assert(~any(any(isnan(W))));
end
```

```
function Y = jacobmultJX(X, vars, meansc)
    Y = zeros(es(n + 1) + 3 * P * (cM + 1), size(X, 2));

    for i = 1:n
        esi = es(i) + 1; esn = es(i + 1) - cM;
        Y(esi:esn, :) = varsJ{i} * X(rvs(i):mve(i), :);
        Y(esi:esn, :) = Y(esi:esn, :) + meshJ{i} * X(vs(n + 1) + 1:vs(n + 1) + 3 * P, :);
        for m = 1:cM
            Y(esi:esn, :) = Y(esi:esn, :) + meshJ{i} * ...
                vars(mvs(i) + m - 1) * X(vs(n + 1) + 1 + m * 3 * P:vs(n + 1) + (m + 1) * 3 * P, :);
        end

        Y(ses(i):see(i) - T(i), :) = Y(ses(i):see(i) - T(i), :) ...
          + diag(silJ{i}(1:T(i), 1)) * X(svs(i):sve(i) - T(i), :);
        Y(ses(i):see(i) - T(i), :) = Y(ses(i):see(i) - T(i), :) ...
          + diag(silJ{i}(1:T(i), 2)) * X(svs(i) + T(i):sve(i), :);
        Y(ses(i) + T(i):see(i), :) = Y(ses(i) + T(i):see(i), :) ...
          + diag(silJ{i}(T(i) + 1:2 * T(i), 1)) * X(svs(i):sve(i) - T(i), :);
        Y(ses(i) + T(i):see(i), :) = Y(ses(i) + T(i):see(i), :) ...
          + diag(silJ{i}(T(i) + 1:2 * T(i), 2)) * X(svs(i) + T(i):sve(i), :);

        D = diag(silJ{i}(2 * T(i) + 1:2 * T(i) + S(i), 1)) + ...
            diag(silJ{i}(2 * T(i) + S(i) + 1:2 * T(i) + 2 * S(i) - 1, 1), 1);
        D(S(i), 1) = silJ{i}(2 * T(i) + 2 * S(i), 1); D(:, fixed_sil_pts{i}) = [];
        Y(ces(i):cee(i), :) = Y(ces(i):cee(i), :) + D * X(svs(i):sve(i) - T(i), :);
        D = diag(silJ{i}(2 * T(i) + 1:2 * T(i) + S(i), 2)) + ...
            diag(silJ{i}(2 * T(i) + S(i) + 1:2 * T(i) + 2 * S(i) - 1, 2), 1);
        D(S(i), 1) = silJ{i}(2 * T(i) + 2 * S(i), 2); D(:, fixed_sil_pts{i}) = [];
        Y(ces(i):cee(i), :) = Y(ces(i):cee(i), :) + D * X(svs(i) + T(i):sve(i), :);

        Y(nes(i):nee(i) - 2 * T(i), :) = Y(nes(i):nee(i) - 2 * T(i), :) ...
          + diag(silJ{i}(2 * T(i) + 2 * S(i) + 1:3 * T(i) + 2 * S(i), 1)) ...
            * X(svs(i):sve(i) - T(i), :);
        Y(nes(i):nee(i) - 2 * T(i), :) = Y(nes(i):nee(i) - 2 * T(i), :) ...
          + diag(silJ{i}(2 * T(i) + 2 * S(i) + 1:3 * T(i) + 2 * S(i), 2)) ...
            * X(svs(i) + T(i):sve(i), :);
        Y(nes(i) + T(i):nee(i) - T(i), :) = Y(nes(i) + T(i):nee(i) - T(i), :) ...
          + diag(silJ{i}(3 * T(i) + 2 * S(i) + 1:4 * T(i) + 2 * S(i), 1)) ...
            * X(svs(i):sve(i) - T(i), :);
        Y(nes(i) + T(i):nee(i) - T(i), :) = Y(nes(i) + T(i):nee(i) - T(i), :) ...
          + diag(silJ{i}(3 * T(i) + 2 * S(i) + 1:4 * T(i) + 2 * S(i), 2)) ...
            * X(svs(i) + T(i):sve(i), :);
        Y(nes(i) + 2 * T(i):nee(i), :) = Y(nes(i) + 2 * T(i):nee(i), :) ...
          + diag(silJ{i}(4 * T(i) + 2 * S(i) + 1:5 * T(i) + 2 * S(i), 1)) ...
            * X(svs(i):sve(i) - T(i), :);
        Y(nes(i) + 2 * T(i):nee(i), :) = Y(nes(i) + 2 * T(i):nee(i), :) ...
          + diag(silJ{i}(4 * T(i) + 2 * S(i) + 1:5 * T(i) + 2 * S(i), 2)) ...
            * X(svs(i) + T(i):sve(i), :);

        Y(es(n + 1) + 1:es(n + 1) + 3 * P, :) = Y(es(n + 1) + 1:es(n + 1) + 3 * P, :) + (xi_0 / n) * ...
            tplatesqrt_3 * vars(vs(n + 1) + 1:vs(n + 1) + 3 * P)' * X(sv(i), :);
        for m = 1:cM
            ms = es(n + 1) + 1 + m * 3 * P; me = es(n + 1) + (m + 1) * 3 * P;
            Y(ms:me, :) = Y(ms:me, :) + tplatesqrt_3 * ...
              (xi_def / n) * vars(vs(n + 1) + 1 + m * 3 * P:vs(n + 1) + (m + 1) * 3 * P)' * X(sv(i), :);
        end

        Y(mes(i):mee(i), :) = X(mvs(i):mve(i), :);
    end

    Y(es(n + 1) + 1:es(n + 1) + 3 * P, :) = Y(es(n + 1) + 1:es(n + 1) + 3 * P, :) + ...
        xi_0 * tplatesqrt_3 * meansc * X(vs(n + 1) + 1:vs(n + 1) + 3 * P, :);
    for m = 1:cM
        ms = es(n + 1) + 1 + m * 3 * P; me = es(n + 1) + (m + 1) * 3 * P;
        Y(ms:me, :) = Y(ms:me, :) + xi_def * tplatesqrt_3 * ...
            meansc * X(vs(n + 1) + 1 + m * 3 * P:vs(n + 1) + (m + 1) * 3 * P, :);
    end
end
```

```
function W = jacobmultJtY(Y, vars, meansc)
    W = zeros(length(vars), size(Y, 2));

    for i = 1:n
        esi = es(i) + 1; esn = es(i + 1) - cM; ms = vs(n + 1) + 1; me = vs(n + 1) + 3 * P;
        W(rvs(i):mve(i), :) = varsJ{i}' * Y(esi:esn, :);
        W(ms:me, :) = W(ms:me, :) + meshJ{i}' * Y(esi:esn, :);
        for m = 1:cM
            ms = vs(n + 1) + 1 + m * 3 * P; me = vs(n + 1) + (m + 1) * 3 * P;
            W(ms:me, :) = W(ms:me, :) + vars(mvs(i) + m - 1) * meshJ{i}' * Y(esi:esn, :);
        end

        W(svs(i):sve(i) - T(i), :) = W(svs(i):sve(i) - T(i), :) ...
          + diag(silJ{i}(1:T(i), 1)) * Y(ses(i):see(i) - T(i), :);
        W(svs(i) + T(i):sve(i), :) = W(svs(i) + T(i):sve(i), :) ...
          + diag(silJ{i}(1:T(i), 2)) * Y(ses(i):see(i) - T(i), :);
        W(svs(i):sve(i) - T(i), :) = W(svs(i):sve(i) - T(i), :) ...
          + diag(silJ{i}(T(i) + 1:2 * T(i), 1)) * Y(ses(i) + T(i):see(i), :);
        W(svs(i) + T(i):sve(i), :) = W(svs(i) + T(i):sve(i), :) ...
          + diag(silJ{i}(T(i) + 1:2 * T(i), 2)) * Y(ses(i) + T(i):see(i), :);

        D = diag(silJ{i}(2 * T(i) + 1:2 * T(i) + S(i), 1)) + ...
            diag(silJ{i}(2 * T(i) + S(i) + 1:2 * T(i) + 2 * S(i) - 1, 1), -1);
        D(1, S(i)) = silJ{i}(2 * T(i) + 2 * S(i), 1); D(fixed_sil_pts{i}, :) = [];
        W(svs(i):sve(i) - T(i), :) = W(svs(i):sve(i) - T(i), :) + D * Y(ces(i):cee(i), :);
        D = diag(silJ{i}(2 * T(i) + 1:2 * T(i) + S(i), 2)) + ...
            diag(silJ{i}(2 * T(i) + S(i) + 1:2 * T(i) + 2 * S(i) - 1, 2), -1);
        D(1, S(i)) = silJ{i}(2 * T(i) + 2 * S(i), 2); D(fixed_sil_pts{i}, :) = [];
        W(svs(i) + T(i):sve(i), :) = W(svs(i) + T(i):sve(i), :) + D * Y(ces(i):cee(i), :);

        W(svs(i):sve(i) - T(i), :) = W(svs(i):sve(i) - T(i), :) ...
          + diag(silJ{i}(2 * T(i) + 2 * S(i) + 1:3 * T(i) + 2 * S(i), 1)) ...
            * Y(nes(i):nee(i) - 2 * T(i), :);
        W(svs(i) + T(i):sve(i), :) = W(svs(i) + T(i):sve(i), :) ...
          + diag(silJ{i}(2 * T(i) + 2 * S(i) + 1:3 * T(i) + 2 * S(i), 2)) ...
            * Y(nes(i):nee(i) - 2 * T(i), :);
        W(svs(i):sve(i) - T(i), :) = W(svs(i):sve(i) - T(i), :) ...
          + diag(silJ{i}(3 * T(i) + 2 * S(i) + 1:4 * T(i) + 2 * S(i), 1)) ...
            * Y(nes(i) + T(i):nee(i) - T(i), :);
        W(svs(i) + T(i):sve(i), :) = W(svs(i) + T(i):sve(i), :) ...
          + diag(silJ{i}(3 * T(i) + 2 * S(i) + 1:4 * T(i) + 2 * S(i), 2)) ...
            * Y(nes(i) + T(i):nee(i) - T(i), :);
        W(svs(i):sve(i) - T(i), :) = W(svs(i):sve(i) - T(i), :) ...
          + diag(silJ{i}(4 * T(i) + 2 * S(i) + 1:5 * T(i) + 2 * S(i), 1)) ...
            * Y(nes(i) + 2 * T(i):nee(i), :);
        W(svs(i) + T(i):sve(i), :) = W(svs(i) + T(i):sve(i), :) ...
          + diag(silJ{i}(4 * T(i) + 2 * S(i) + 1:5 * T(i) + 2 * S(i), 2)) ...
            * Y(nes(i) + 2 * T(i):nee(i), :);

        W(sv(i), :) = W(sv(i), :) + (xi_0 / n) * vars(vs(n + 1) + 1:vs(n + 1) + 3 * P) * ...
            tplatesqrt_3' * Y(es(n + 1) + 1:es(n + 1) + 3 * P, :);
        for m = 1:cM
            ms = es(n + 1) + 1 + m * 3 * P; me = es(n + 1) + (m + 1) * 3 * P;
            W(sv(i), :) = W(sv(i), :) + (xi_def / n) * ...
              vars(vs(n + 1) + 1 + m * 3 * P:vs(n + 1) + (m + 1) * 3 * P) * tplatesqrt_3' * Y(ms:me, :);
        end

        W(mvs(i):mve(i), :) = W(mvs(i):mve(i), :) + Y(mes(i):mee(i), :);
    end

    W(vs(n + 1) + 1:vs(n + 1) + 3 * P, :) = W(vs(n + 1) + 1:vs(n + 1) + 3 * P, :) + ...
        xi_0 * meansc * tplatesqrt_3 * Y(es(n + 1) + 1:es(n + 1) + 3 * P, :);
    for m = 1:cM
         ms = es(n + 1) + 1 + m * 3 * P;  me = es(n + 1) + (m + 1) * 3 * P;
        sms = vs(n + 1) + 1 + m * 3 * P; sme = vs(n + 1) + (m + 1) * 3 * P;
        W(sms:sme, :) = W(sms:sme, :) + xi_def * tplatesqrt_3 * meansc * Y(ms:me, :);
    end
end
```