Implementing the Automatic Generation of 3D Statistical Shape Models with ITK

Release 0.00

Tobias Heimann[1], Ipek Oguz[2], Ivo Wolf[1], Martin Styner[2] and Hans-Peter Meinzer[1]

July 10, 2006

[1]Div. Medical and Biological Informatics, German Cancer Research Center,
t.heimann@dkfz-heidelberg.de
[2]University of North Carolina at Chapel Hill, ipek@cs.unc.edu

Abstract

Statistical Shape Models are a popular method for segmenting three-dimensional medical images. To obtain the required landmark correspondences, various automatic approaches have been proposed. In this work, we present an improved version of minimizing the description length (MDL) of the model. To initialize the algorithm, we describe a method to distribute landmarks on the training shapes using a conformal parameterization function. Then, we introduce a novel procedure to modify landmark positions locally without disturbing established correspondences. We employ a gradient descent optimization to minimize the MDL cost function, speeding up automatic model building by several orders of magnitude when compared to the original MDL approach. The necessary gradient information is estimated from a singular value decomposition, a more accurate technique to calculate the PCA than the commonly used eigendecomposition of the covariance matrix. In this work, we first present a basic version where spatial locations are used in the MDL cost function; next, we introduce an extended version where any combination of features can be used as a metric. As an example application, we present results based on local curvature measurements. Finally, we present results for synthetic and real-world datasets demonstrating the efficiency of our procedures and give details about the implementation using the Insight Toolkit (ITK).

Contents

1 Introduction

Since their introduction by Cootes et al. [4], Active Shape Models (ASMs) and statistical shape methods in general have become popular tools for automatic segmentation of medical images. The main challenge

of the approach is the point correspondence problem in the model construction phase: On every training sample for the shape model, landmarks have to be placed in a consistent manner. While manual labeling is a time-consuming but feasible solution for 2D models when using only a limited number of landmarks, it is highly impractical in the 3D domain: Not only is the required number of landmarks higher than in the 2D case, but it also becomes increasingly difficult to identify and pinpoint corresponding points, even for experts.

Several automated methods to find the correspondences in 3D have been proposed. Brett and Taylor [3] use a pairwise corresponder based on a symmetric version of the ICP algorithm. All training shapes are decimated to generate sparse polyhedral approximations and then merged in a binary tree, which is used to propagate landmark positions. Shelton [20] measures correspondence between surfaces in arbitrary dimensions by a cost function which is composed of three parts representing Euclidean distance, surface deformation and prior information. The function is minimized using a multi-resolution approach that matches highly decimated versions of the meshes first and iteratively refines the results. Paulsen and Hilger [19] match a decimated template mesh to all training shapes using thin plate spline warping controlled by a small set of manually placed anatomic landmarks. The resulting meshes are relaxed to fit the training shapes by a Markov random field regularization. Another approach based on matching templates is presented by Zhao and Teoh [23]: They employ an adaptive-focus deformable model to match each training shape to all others without the need for manually placed landmarks. The shape yielding the best overall results in this process is subsequently used to determine point correspondences, enhanced by a ”bridge-over” procedure for outliers. Another approach presented by Meier in [16] explores using local curvature measurements for establishing pairwise correspondence. The cost function is based on curvature (C) and shape index (S) metrics defined in [15].

A common characteristic of these methods is that they base their notion of correspondence on general geometric properties, e.g. minimum Euclidean distance and low distortion of surfaces. A different approach is presented by Davies et al. [7] who propose to minimize a cost function based on the minimum description length of the resulting statistical shape model. In a recent comparison [21], this approach has shown to be superior to other correspondence methods. However, the optimization of the MDL criterion for 3D shapes is complex to implement and computationally expensive. In [13], we presented an optimized procedure for minimizing the description length which is easier to implement and more efficient than the original approach. In this paper, we describe an implementation of our algorithm using the pipeline architecture of the open source toolkit ITK. We will start by reviewing the theory of the algorithm and present some results before we present the ITK implementation in Sect. 6.

2 Fundamentals

2.1 Statistical Shape Models

The most popular kind of shape models uses point distribution models (PDMs), which represent each d-dimensional training sample as a set of n landmarks. For every sample, landmark positions are defined by a single vector $\mathbf{x}$, storing the coordinates for landmark i at (x_i, x_{i+n}, x_{i+2n}). The vectors of all training samples form the columns of the landmark configuration matrix $\mathbf{L}$. Applying principal component analysis (PCA) to this matrix delivers the principal modes of variation $\mathbf{p}_m$ in the training data. Restricting the model to the first c modes, all valid shapes can be approximated by the mean shape $\overline{\mathbf{x}}$ and a linear combination of

displacement vectors:

$$\mathbf{x} = \bar{\mathbf{x}} + \sum_{m=1}^{c} y_m \mathbf{p}_m \tag{1}$$

Cootes used an eigenvector decomposition of the covariance matrix of $\mathbf{L}$ to calculate the PCA [4], a method commonly employed for this purpose. However, the same results can also be achieved by a singular value decomposition (SVD), which is numerically more stable and thus more accurate when the covariance matrix is ill-conditioned [14].

Theorem 1 *Any $m \times n$ real matrix $\mathbf{A}$ with $m \geq n$ can be written as the product*

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathbf{T}} \tag{2}$$

where $\mathbf{U}$ and $\mathbf{V}$ are column orthogonal matrices of size $m \times n$ and $n \times n$, respectively, and $\mathbf{D}$ is a $n \times n$ diagonal matrix. Then $\mathbf{U}$ holds the eigenvectors of the matrix $\mathbf{A}\mathbf{A}^{\mathbf{T}}$ and $\mathbf{D}^{\mathbf{2}}$ the corresponding eigenvalues.

Without calculating the covariance matrix, the PCA can thus be obtained by the SVD of the matrix $\mathbf{A} = \frac{1}{\sqrt{s-1}}(\mathbf{L} - \bar{\mathbf{L}})$, where s is the number of samples and $\bar{\mathbf{L}}$ a matrix with all columns set to $\bar{\mathbf{x}}$. In addition to the increased accuracy, the matrices $\mathbf{U}$ and $\mathbf{V}$ allow calculating gradient information for the eigenvalues which we will use during the optimization stage of the model-building process.

2.2 Correspondence by minimizing description length

A prerequisite for statistical shape models is a set of landmark points located at corresponding positions on all training shapes. To quantify this correspondence, the MDL approach introduced by Davies et al. [6] defines a cost function F which is based on the minimum description length of the generated model. In this work, we use a simplified version of the MDL as proposed by Thodberg [22], where F is defined as:

$$F = \sum_m \mathcal{L}_m \quad \text{with} \quad \mathcal{L}_m = \begin{cases} 1 + \log(\lambda_m/\lambda_{\text{cut}}) & \text{for } \lambda_m \geq \lambda_{\text{cut}} \\ \lambda_m/\lambda_{\text{cut}} & \text{for } \lambda_m < \lambda_{\text{cut}} \end{cases} \tag{3}$$

This formulation features one free parameter λ_{cut} which represents the expected noise in the training data. Since all shapes are rescaled to produce a mean shape with RMS radius $r = 1/\sqrt{n}$ for the PCA, the optimal value for λ_{cut} depends on the original average radius of the training shapes $\bar{r}$:

$$\lambda_{\text{cut}} = \left(\frac{\sigma}{\bar{r}}\right)^2 , \tag{4}$$

where σ is the standard deviation of noise in the training data. In coherence with the voxel quantization error, Thodberg choses $\sigma = 0.3$ and uses $\bar{r} = 100$ in all his experiments. While we adopt the same σ-value, we modify $\bar{r}$ depending on the resolution of the images from which the training shapes are extracted.

2.3 Using different metrics for correspondence

The basic MDL method uses 3D spatial location information as its metric. However, it is also possible to use any other local metrics, and minimize the model description length with respect to these. A good example of such features that can be used for establishing correspondence is local curvature metrics.

In this work, we present results using the local curvature metrics presented in Koenderink [15], namely, the shape index S and the curvedness C. C and S can be computed as functions of the two principal curvatures of the surface. They basically are equivalent to a polar representation of the principal curvatures κ_1 and κ_2.

$$C = \frac{2}{\pi} ln \sqrt{(\kappa_1^2 + \kappa_2^2)/2} \tag{5}$$

$$S = -\frac{2}{\pi} arctan \frac{\kappa_1 + \kappa_2}{\kappa_1 - \kappa_2} \tag{6}$$

C and S improve the curvature measurement by decoupling the size and shape aspects of the curvature. C describes how curved an object is, and is closely related to the size. S, on the other hand, is indicating the shape of the surface in terms of concaveness and convexness. This pair of metrics is very suitable for measuring correspondence of two surfaces, since they provide a means of measuring shape in a very intuitive way.

It should be noted that, even though we only present results using C and S as metrics in this work, our implementation provides complete flexibility in the choice of features to be used. This is achieved by letting the user provide the number of features per point and feature values, without any constraints. The feature values should be computed offline and stored in a feature file for each object in the population.

3 Mesh Parameterization

To define an initial set of correspondences and a means of manipulating them efficiently, we need a convenient parameter domain for our training shapes. For closed 2D objects, the natural choice for this parameter domain is the arc-length position on the contour: Choosing an arbitrary starting point and normalizing the total arc-length to 1, all positions on the contour (i.e. all potential landmark positions) can be described by a single parameter $p \in [0..1]$.

In order to minimize complexity for the parameterization of 3D shapes, we will restrict the discussion to closed two-manifolds of genus 0 (i.e. surfaces without holes and self-intersections). Objects of this class are topologically equivalent to a sphere and most shapes encountered in medical imaging are of this type(e.g. liver, kidneys and lungs). The task is to find a one-to-one mapping which assigns every point on the surface of the mesh a unique position on the unit sphere, described by two parameters longitude $\theta \in [0..2\pi]$ and latitude $\phi \in [0..\pi]$.

The mapping of an arbitrary shape to a sphere inevitably introduces some distortion. There are a number of different approaches which attempt to minimize this distortion, typically preserving either local angles or facet areas while trying to minimize distortions in the other. An overview of recent work on this topic can be found in [9].

For an initial parameterization, Davies uses diffusion mapping, a simplified version of the spherical harmonics method described by Brechbühler [2]. For our optimization strategy (Sect. 4), an angle preserving method is also suitable: Moving neighboring points on the parameterization sphere in a specific direction, we expect the corresponding landmarks on the training shape to move in a coherent direction as well. This behavior is guaranteed by conformal mapping functions, transformations that preserve local angles.

3.1 Creating a conformal mapping

Definition 1 *Each training sample for the statistical shape model is represented as a triangulated mesh* $K = (V,E)$ *with vertices* $u,v \in V$ *and edges* $[u,v] \in E$. *The vertex positions are specified by* $\mathbf{f} : V \to R^3$, *an embedding function defined on the vertices of K. A second function* $\omega : V \to R^3$ *specifies the coordinates as mapped on the unit sphere,* $\forall v \in V : |\omega(v)| = 1$.

Gu et al. present a variational method to create a conformal parameterization in [12]. From an initial Gauss map, where $\omega(v)$ represents the normal vector of v, they use a gradient descent optimization to minimize the string energy of the mesh, defined as:

$$\mathcal{E}(K,\omega) = \sum_{[u,v]\in E} k_{u,v} \|\omega(u) - \omega(v)\|^2 \tag{7}$$

Minimizing the string energy with all edge weights $k_{u,v}$ set to 1 yields the barycentric mapping, where each vertex is positioned at the center of its neighbors. Subsequently, a conformal mapping can be obtained using edge weights depending on the opposing angles α, β of the faces adjacent to $[u,v]$ as in:

$$k_{u,v} = \frac{1}{2}(\cot\alpha + \cot\beta) \tag{8}$$

During the optimization process, all vertices must constantly be projected back onto the sphere by $\omega(u) = \omega'(u)/|\omega'(u)|$. The formal correctness of this approach was later proved in [11].

3.2 Spherical harmonics

An alternative method for obtaining a spherical parametrization of the input objects is to use spherical harmonics functions. In summary, the SPHARM description is a hierarchical, global, multi-scale boundary description that can only represent objects of spherical topology [2]. The spherical parameterization is computed via optimizing an equal area mapping of the 3D voxel mesh onto the sphere and minimizing angular distortions. The basis functions of the parameterized surface are spherical harmonics. Each individual SPHARM description is composed of a set of coefficients, weighting the basis functions. Truncating the spherical harmonic series at different degrees results in object representations at different levels of detail. SPHARM is a smooth, accurate fine-scale shape representation, given a sufficiently high representation level. Based on a uniform icosahedron-subdivision of the spherical parameterization, we obtain a Point Distribution Model (PDM).

Spherical harmonic basis functions Y_l^m, $-l \le m \le l$ of degree l and order m are defined on $\theta \in [0;\pi] \times \phi \in [0;2\pi)$ by the following definitions:

$$Y_l^m(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(\cos\theta)\, e^{im\phi} \tag{9}$$

$$Y_l^{-m}(\theta,\phi) = (-1)^m Y_l^{m*}(\theta,\phi)\,, \tag{10}$$

where Y_l^{m*} denotes the complex conjugate of Y_l^m and P_l^m the associated Legendre polynomials

$$P_l^m(w) = \frac{(-1)^m}{2^l\, l!}(1-w^2)^{\frac{m}{2}} \frac{d^{m+l}}{dw^{m+l}}(w^2-1)^l. \tag{11}$$

Our ITK-compatible implementation for generating the spherical harmonics representation of a surface, as well as other classes for handling spherical harmonics representation, is publicly available at UNC Neurolib repository (www.ia.unc.edu/dev/). These classes are not provided as part of this work, since the initial parametrization is assumed to be precomputed and stored.

3.3 Mapping landmarks

Following the preceding sections, the parameterization is defined by a spherical mesh with the same topology as the training sample. In order to obtain the 3D position for an arbitrary landmark at the spherical coordinates (θ, ϕ), which is generally not a vertex, we have to find the intersection between a ray from the origin to (θ, ϕ) and the parameterization mesh. Since mapping landmarks is the most computationally expensive part of the model-building process, an intelligent search strategy of ordering the triangles according to the likelihood of ray intersection speeds up the algorithm considerably. Intersected triangle indices for each landmark are cached and, in the case of a cache miss, neighboring triangles are given priority when searching for the ray intersection. To test a triangle for intersection, we use the method described in [17], which conveniently produces the barycentric coordinates of the intersection point. The same coordinates used on the respective triangle of the training mesh yield the final landmark position.

4 Optimizing Landmark Correspondences

With an initial conformal parameterization ω_i for each training sample i, we can acquire the necessary landmarks by mapping a set of spherical coordinates to each shape. To optimize the point correspondences with the MDL criterion, two possibilities are available: We can either change the individual ω_i and maintain a fixed set of global landmarks or modify individual landmark sets Ψ_i.

In this work, we opted for the first alternative, which has the advantage that the correspondence is valid for any set of points placed on the unit sphere. Therefore, it is possible to alter number and placement of landmarks on the unit sphere at any stage of the optimization, e.g. to better adapt the triangulation to the training shapes. Moreover, we do not need to worry about the correct ordering of landmarks: Since the valid set on the unit sphere is fixed, ensuring a one-to-one mapping to the training shapes is sufficient.

4.1 Re-parameterization

To modify the individual parameterizations in an iterative optimization process, we need a transformation function of the type $\omega' = \Phi(\omega)$. In [7], Davies et al. use symmetric theta transformations for that purpose: Employing a wrapped Cauchy kernel with a certain width and amplitude, landmarks near the kernel position are spread over the sphere, while landmarks in other regions of the surface are compressed. By accumulating the effects of thousands of kernels at different positions, arbitrary parameterizations can be created.

While this re-parameterization method produces the required effect, it is an inefficient means of modifying surface parameterizations. The main disadvantage is that it is a global modification, i.e. adding one new kernel modifies all landmark positions on the object. Intuitively, it would be desirable to keep established landmark correspondences stable. Therefore, we suggest a new method for modifying parameterization functions based on kernels with strictly local effects.

We will assume that we know a principal direction $(\Delta\theta, \Delta\phi)$ in which the vertices of a local neighborhood on the parameterization mesh should move to improve landmark correspondences. Then we define a Gaussian

envelope function to change each spherical coordinate by $c(x,\sigma)\cdot(\Delta\theta,\Delta\phi)$ with

$$c(x,\sigma)=\begin{cases} e^{\frac{-x^2}{2\sigma^2}}-e^{\frac{-(3\sigma)^2}{2\sigma^2}} & \text{for } x<3\sigma \\ 0 & \text{for } x\geq 3\sigma \end{cases} \tag{12}$$

The variable x denotes the Euclidean distance between the center of the kernel and the specific vertex of the parameterization mesh, while σ specifies the size of the kernel. The movements are cut off at 3σ to limit the range and keep the modification local. During the course of the optimization, σ is decreased to optimize larger regions at the beginning and details at the end. Three examples for possible kernel configurations with different σ-values are shown in Fig. 1.

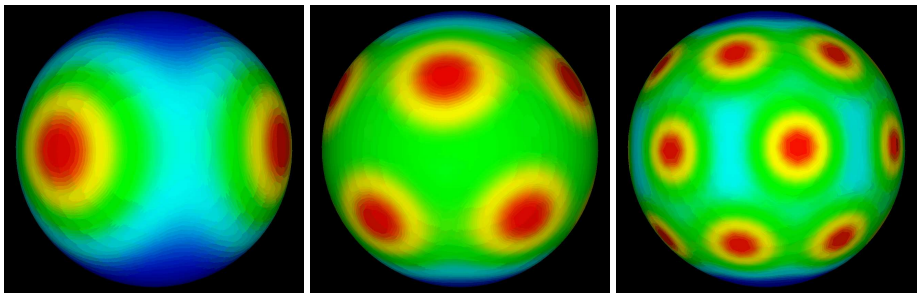

Figure 1: Kernel configurations for σ values of 0.4, 0.3 and 0.2. Red colors mark regions with large vertex movements, blue ones those with no modification.

The proposed method of modification does not work if a kernel includes one of the poles of the spherical parameterization mesh ($\phi=0$ or $\phi=\pi$) because vertices would all move either toward or away from this point, depending on $\Delta\phi$. Nevertheless, the positions of the different kernels have to change in the course of the optimization in order to guarantee an equal treatment for all vertices of the parameterization mesh. This limitation is overcome by defining specific kernel configurations as shown in Fig. 1, which do not cover the pole sections of the sphere. By keeping these configurations fixed and instead rotating all parameterizations and the global landmark collection by a random rotation matrix, the relative kernel positions are changed without touching a pole. The random rotation matrices for these operations are acquired using the method described in [1].

4.2 Calculating MDL gradients

Given a kernel at a certain position, we need the direction $(\Delta\theta,\Delta\phi)$ for the movement which minimizes the cost function. Since all modifications of the parameterization change landmark positions on the training sample, the first step is to quantify the effect landmark movements have on the MDL value. As shown in [8], the work of Papadopoulo and Lourakis on estimating the Jacobian of the SVD [18] can be used for that purpose, calculating the gradients of the MDL objective function with respect to individual landmarks.

The calculation of the singular value derivatives does not add a significant computational overhead. Given the centered and un-biased landmark configuration matrix $\mathbf{A}$ from Sect. 2.1, the derivative for the m-th singular value d_m is calculated by:

$$\frac{\partial d_m}{\partial a_{ij}}=u_{im}\cdot v_{jm} \tag{13}$$

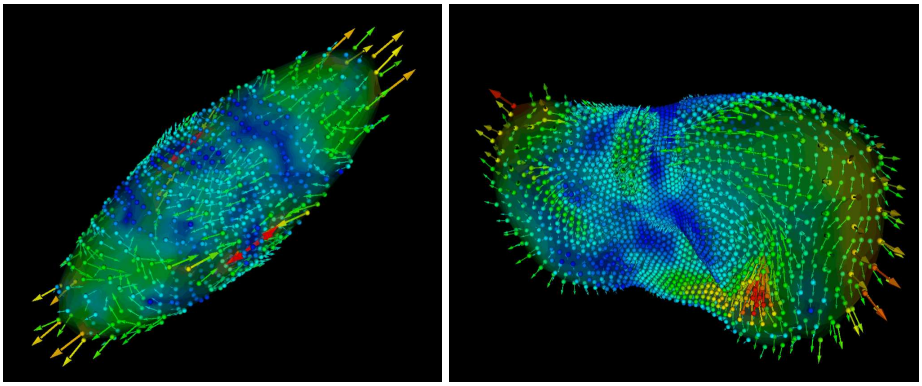

Figure 2: Gradients of the MDL cost function visualized for two sample shapes. The value of the directional derivative is color-coded ranging from blue for weak gradients to red for the strongest gradients.

The scalars u_{im} and v_{jm} are elements of the matrices $\mathbf{U}$ and $\mathbf{V}$ from (2). Since our MDL cost function uses $\lambda_m = d_m^2$, we can derive the MDL gradients as

$$\frac{\partial F}{\partial a_{ij}} = \sum_m \frac{\partial \mathcal{L}_m}{\partial a_{ij}} \quad \text{with} \quad \frac{\partial \mathcal{L}_m}{\partial a_{ij}} = \begin{cases} 2u_{im}v_{jm}/d_m & \text{for } \lambda_m \geq \lambda_{\text{cut}} \\ 2d_m u_{im} v_{jm}/\lambda_{\text{cut}} & \text{for } \lambda_m < \lambda_{\text{cut}} \end{cases} \tag{14}$$

This derivation yields a 3D gradient for every landmark, revealing the influence of its movements on the cost function. Two examples of the resulting gradient fields are visualized in Fig. 2.

4.3 Putting it all together

The final step is to transform the calculated gradient fields into optimal kernel movements $\mathbf{k} = (\Delta\theta, \Delta\phi)$ on the parameterization mesh. Using the chain rule, we get:

$$\frac{\partial F}{\partial \mathbf{k}} = \frac{\partial F}{\partial a_{ij}} \frac{\partial a_{ij}}{\partial \mathbf{k}} \tag{15}$$

We use finite differences to estimate the surface gradients $\partial a_{ij}/\partial \mathbf{k}$.

Both Davies [5] and Thodberg [22] describe cases in which the MDL optimization can lead to landmarks piling up in certain regions or collapsing to a point. Davies keeps one shape as a master example with fixed landmarks to prevent this effect while Thodberg suggests adding a stabilizing term to the cost function. Since we have never observed the problematic behavior with our new re-parameterization, we do not employ any of these methods.

In addition to modifying the mapping functions ω_i by re-parameterization, other variables which influence landmark positions can be included in the optimization. The rotation of each mapping ω_i determines the position of the first landmark on the training shape and the relative orientation of all others. By calculating gradients for rotating the parameterization mesh around the three Euclidean axes and using those instead of the surface gradients $\partial a_{ij}/\partial \mathbf{k}$ in (15), we have an efficient method to optimize this variable.

Table 1: The collection of datasets used for the evaluation.

	Cuboids	Ellipsoids	Livers
Origin	synthetic	synthetic	clinical
Mean size (radius in voxels)	100	100	70
Number of samples	20	20	21
Perceived sample variance	low	medium	high
Sample complexity (# vertices)	486	962	1500–2000
Model complexity (# landmarks)	642	642	2562

Table 2: Cost function values F and number of used iterations I for all datasets. Times are given in hours and minutes.

	Cuboids			Ellipsoids			Livers		
State	Time	I	F	Time	I	F	Time	I	F
Initial values	0:00	0	15.86	0:00	0	17.47	0:00	0	X
After optimization	0:11	900	10.96	0:13	1400	13.13	X	X	X

5 Results

5.1 Spatial location based MDL

Datasets

As previously noted, the basic method uses the 3D spatial location of vertices for establishing correspondence. We tested the presented method on two synthetic and one real-life dataset. Synthetic data has the advantage that the global minimum of the cost function is known, since it can be calculated from the correspondences inherent for generated data. A tabular description of all employed datasets is given in Tab. 1.

Optimization of Point Correspondences

Using the algorithm presented herein, we created statistical shape models for all three datasets on an Intel Dual Xenon 3.0GHz platform. The results of the optimization are summarized in Tab. 2. An extensive evaluation of the created shape models including a comparison with the original MDL approach was conducted in [13]. It turned out that our new algorithm is several orders of magnitude faster and delivers signi`fi`cantly better models than the original approach.

5.2 Local curvature based MDL

The extended version of the presented method can use any number of local features for establishing correspondence. The feature values at each location are provided in input `fi`les. Here, we present results of an experiment where we used the previously presented local curvature metrics C and S as our features. Figures 3 and 4 show the results of this optimization, visualized such that corresponding locations across the population are colored in the same way. Figure 3 shows the ϕ value correspondence and Figure 4 shows the θ value correspondence, where ϕ and θ are the usual spherical coordinates.

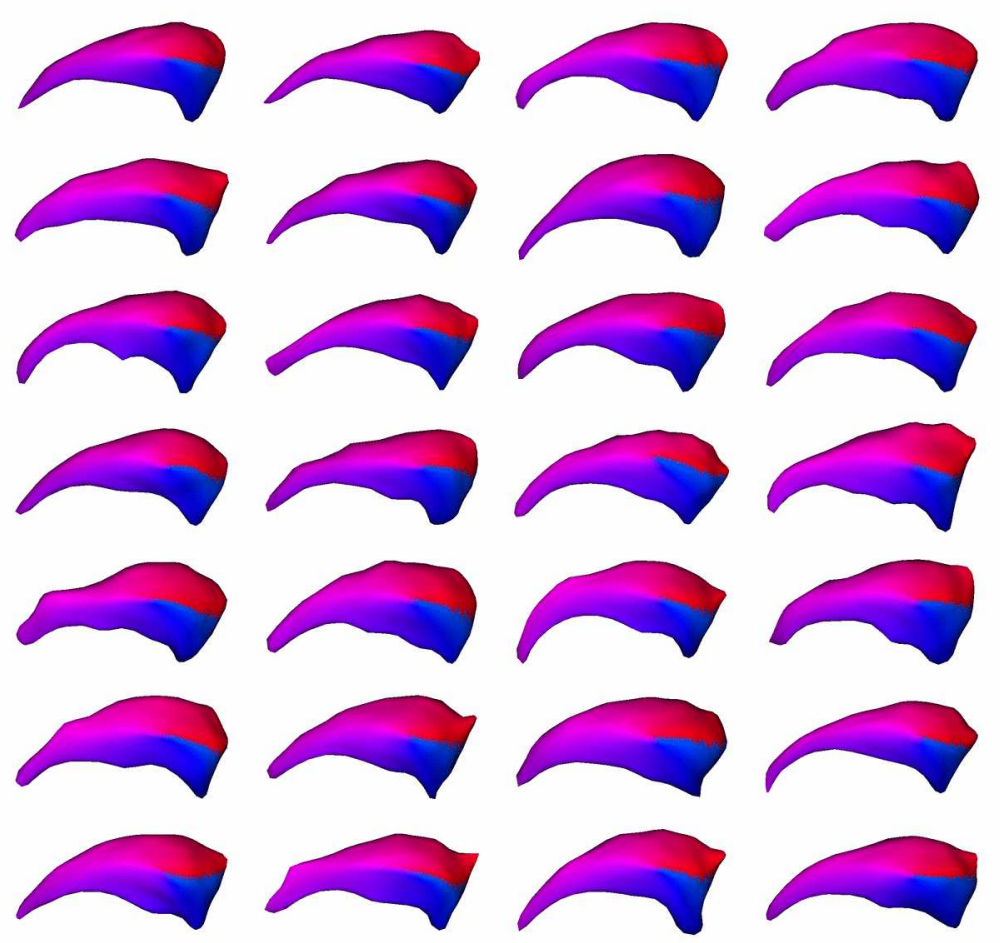

Figure 3: The ϕ values of our population after curvature based correspondence is run. Similar colors across the objects show corresponding ϕ values on each object.

Note that even though we present results using only C and S metrics, our tool allows the usage of any desired optimization metric, or any combination thereof. This is achieved by making the correspondence based on input read through a file, and not internal computations. This provides great flexibility and enables exploring various shape metrics and inspecting the quality of the correspondence they imply, without even modifying the code.

6 Implementation

Although the proposed algorithm is easier to implement than the original MDL optimization, it is still a challenge. One of the earliest problems encountered was that ITK, while offering a large variety of 2D and 3D image filters, provides only very limited mesh support. Most of the functionality necessary for parameterizing meshes — beginning with efficient access to vertices, edges and faces — had to be implemented from scratch in diverse subclasses of `itk::Mesh`. Consequently, there was a lot of work to do apart from designing the core components of the algorithm. An overview of how these classes act together in the algorithm for automatic model building is given in Fig. 5. Detailed information on all implemented classes is presented in Appendix A.

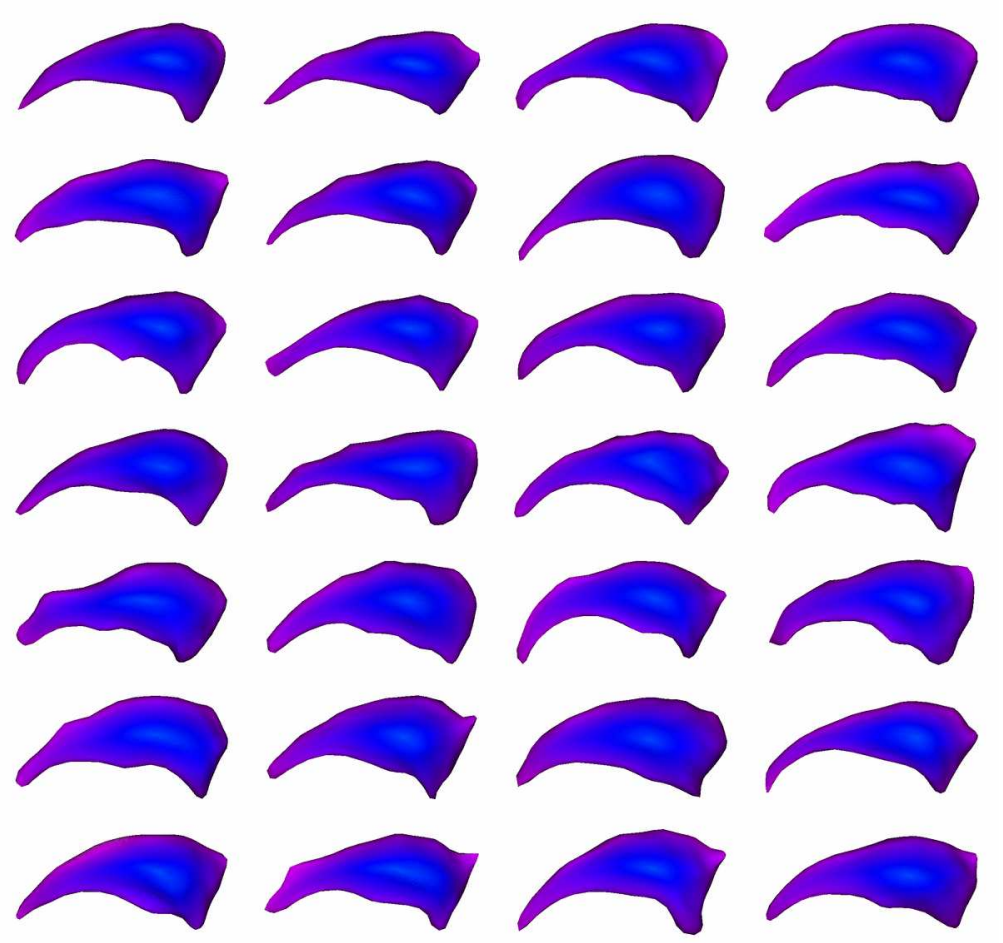

Figure 4: The θ values of our population after curvature based correspondence is run. Similar colors across the objects show corresponding θ values on each object.

A `main()` function is provided along with these classes as a ready to use tool. The only parameters to this tool are an input list `file`, a landmark `file`, and a model radius. The input list `file` is a simple text `file` including the paths for all the input mesh `file`s representing the input objects in the population. For each object, there should be separate files containing the vertices, the faces, the parametrization, and the features (if the features rather than the spatial locations are to be used). Each of these files should have the same name, but different extensions: *.pts* for the vertices, *.fce* for the faces, *.par* for the parametrization, and *.txt* for the features. More information about the file formats can be found in the Appendix A.14.

The landmark `file` is a separate mesh `file` that represents the locations that we want to include in the evaluation of the MDL cost function. This can be the same as one of the objects in the population if the resolution of the mesh is suitable. A typical usage, however, would have high-resolution input meshes with a coarser landmark mesh, so that the optimization proceeds faster. The last parameter, the model radius, is a variance threshold that is used to determine the amount of noise in the data.

A detailed overview of how the `main()` function works is useful to demonstrate how the various classes should be put together. Initially, an instance of the `StatisticalShapeModel3DCalculator` class is created, and it is provided with a cost function, which is an instance of the `SimplifiedMDLCostFunction` class. Note that one can either choose the `StatisticalShapeModel3DCalculator` class itself and use spatial locations as a metric, or use the subclass `StatisticalShapeModel3DCalculatorWithFeatures` and use arbitrary local features. Next, the input meshes and the landmark mesh are loaded. If a parametrization `file` is not already provided along with the input meshes, a suitable initial parametrization is computed, either via conformal spherical parametrization or via spherical harmonics basis functions (the former method is illustrated in the provided `main()` function). Any other method that generates a spherical parametrization can be used as well. The resulting instances of the `SphericalParametrizedTriangleMesh` class are then provided to the `StatisticalShapeModel3DCalculator`. After the `StatisticalShapeModel3DCalculator` is updated, all that remains to do is to output the `final` versions of the meshes. Across the population of output meshes, the points with the same (ϕ, θ) values will be corresponding.

7 Conclusions

We presented our automatic statistical shape model building method and its implementation in the open source ITK framework. It offers an `efficient`, robust and versatile approach to automatic model building that should further propagate the use of 3D shape models in clinical practice. To represent more complex shapes (e.g. brain ventricles), the mesh surface could be cut and parameterized over multiple domains instead of a single sphere.

Future research will investigate how far the established correspondences can be used to reorganize landmarks after the optimization in order to represent the geometry of the model optimally with a minimum number of points. Additionally, the stability of our re-parameterization method against landmark collapse has to be `verified` using a larger number of test datasets.

8 Acknowledgements

We are thankful to Guido Gerig and Steve Pizer (UNC Chapel Hill), for highly valuable discussions in regard to shape correspondence and surface curvature. The caudate segmentations used in this manuscript were provided by Jim Levitt and Martha Shenton, (Dept. Psychiatry, VA Boston Healthcare System, Brockton

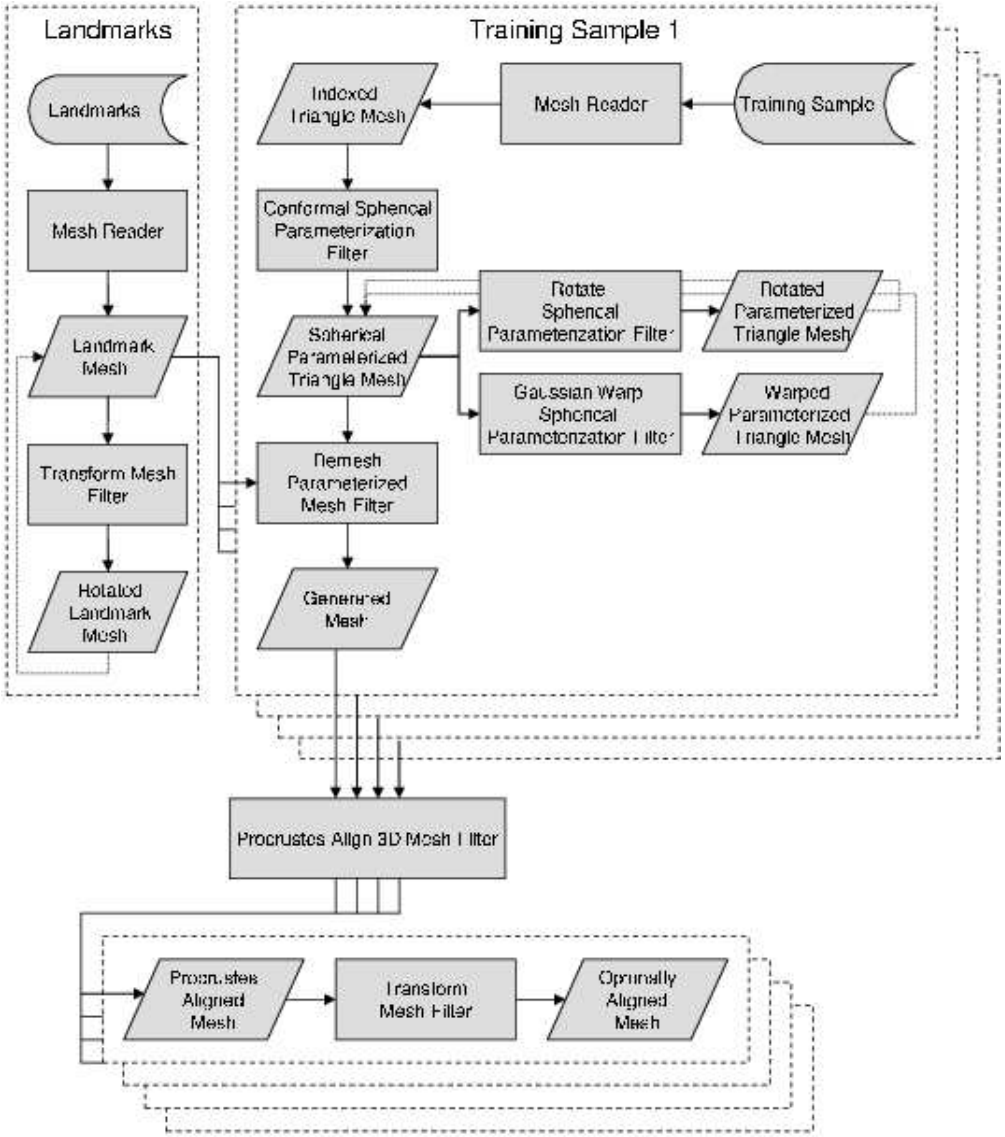

Figure 5: An overview of the pipeline involved in the algorithm for automatic model building. Each training sample is read from disk and parameterized conformally. Using a landmark mesh which is also read from disk, shapes with the same number of vertices are created. These are aligned by a Generalized Procrustes matching and scaled to tangent size. In each optimization step, all parameterizations are modified by the Gaussian warp filter and the results written back to the original data (dotted line). Subsequently, landmarks and parameterizations are rotated with the same transform (i.e. landmark positions on the generated meshes do not change), again overwriting the original values.

Division, Harvard, Boston). This research is supported by the NIH Roadmap for Medical Research Grant U54 EB005149-01, UNC Neurodevelopmental Disorders Research Center HD 03110, and the NIH NIBIB grant P01 EB002779.

A Appendix: Class implementations

In the following, we present all implemented classes one by one.

A.1 IndexedTriangleMesh

The `itk::IndexedTriangleMesh` class represents a three-dimensional, two-manifold mesh consisting exclusively of triangle cells. It derives from the `itk::Mesh` class and hence inherits all the common functionality related to points and cells. It was designed to allow fast and easy access to adjacency information of points, edges and faces. This information is precalculated and stored in the overwritten `BuildCellLinks()` method. In order to stay up-to-date, this method has to be called after all topological modifications to the mesh. In addition to indexing points, the class also indexes faces (i.e. triangles) and edges. Although edges do have a direction, an edge shared by two faces is only stored once under a single index. The index data type can be specified as a template parameter, for meshes with less than 65,536 faces, edges and vertices, unsigned short is recommended to save memory. There is an assignment operator for an `itk::IndexedTriangleMesh`, which can be used e.g. in filters to initialize the output with the input. It is important to note that the operator only copies the pointers (i.e. performs a shallow copy) for all `itk::Mesh` information, while all adjacency information is copied by value (deep copy).

Point related functionality.

In addition to the `GetPoint()` method of the `itk::Mesh` class that copies point information to a supplied pointer, there is an overloaded version that returns a reference to the point. For a given edge, the two points forming it can be queried with `GetPointIndexForEdge()`. If one point of the edge is known (but it is not clear whether it has the local index 0 or 1), the other point can be queried by `GetConnectedPointIndex()`. The same way, the three points forming a triangle can be accessed using `GetPointIndexForFace()`. If one edge of the face and thus two points forming it are known, the third point can be queried using `GetMissingPointIndex()`.

Edge related functionality.

The number of edges in the mesh can be queried with `GetNumberOfEdges()`. All edges are indexed from 0 to `GetNumberOfEdges()-1`. An edge is represented as a vector and can be accessed by the `GetEdge()` method, its length by `GetEdgeLength()`. The number of edges connected to a specific point can be queried with `GetNumberOfEdgesForPoint()`, their indices with `GetEdgeIndexForPoint()`. The following code calculates the average edge length for a specific point:

```
IndexType pointId = 0;
CoordRepType avgLength = 0;
for (IndexType i=0; i<GetNumberOfEdgesForPoint( pointId ); i++)
{
```

```
  avgLength += GetEdgeLength( GetEdgeIndexForPoint( pointId, i ) );
}
avgLength /= GetNumberOfEdgesForPoint( pointId );
```

The three edges forming a specific triangle can be accessed by `GetEdgeIndexForFace()`. To test whether two points are connected by an edge, the method `GetConnectingEdgeIndex()` can be used.

Face related functionality.

The number of faces in the mesh can be queried with `GetNumberOfFaces()`. All faces are indexed from 0 to `GetNumberOfFaces()-1`. The number of faces connected to a specific point can be queried with `GetNumberOfFacesForPoint()`, their indices with `GetFaceIndexForPoint()`. Two faces count as adjacent if they share at least one common point. This means that a face can (and mostly will have) more than the obvious three adjacent faces sharing the three edges. The exact number can be queried using `GetNumberOfAdjacentFaces()`, the respective face indices using `GetAdjacentFaceIndex()`.

A.2 ParameterizedTriangleMesh

The `itk::ParameterizedTriangleMesh` class is the base class for a surface parameterization of a three-dimensional, two-manifold mesh consisting exclusively of triangle cells. It derives from the `itk::IndexedTriangleMesh` class and extends it by adding methods for mapping between parameter and object space. Although coordinates in parameter space can be expressed using only two values (since the mesh is a two-manifold), all mapping methods use the inherited, three-dimensional `PointType` to pass parameter coordinates. How to convert from this `PointType` to the two-dimensional parameter coordinates is up to the individual subclasses. The method `MapCoordinates()` maps coordinates from parameter space to object space. Object space coordinates are represented as a face index and barycentric coordinates inside this face. If the face the mapping will lead to is already known or suspected, the method `CoordinatesInFace()` can be used to find the exact barycentric coordinates. To display the parameterization graphically in a 2D-image (e.g. for purposes of texture mapping), usually a number of different patches (images) are used to minimize distortion. The number of patches can be assessed using `GetNumberOfPatches()`. The patch which should be used to display a certain face with minimal distortion can be queried with `GetPatchIndexForFace()`. Once the patch is known, the two methods `MapParameterizationToPatch()` and `MapPatchToParameterization()` can be used to map coordinates from one domain to the other. Patch coordinates always lie in the range of [0..1]. `UpdateParameterization()` can be used to copy the parameterization from another `itk::ParameterizedTriangleMesh` (usually one modified by a filter operation). Subsequently, `GetParameterizationModified()` reveals if the parameterization for a specific point was changed during this process. The method `SetParameterizationModified()` allows filters to set the modified-flag for a specific point or for all points at once. All presented methods are declared virtual and — with the exception of the `Get/SetParameterizationModified()` methods — have to be implemented in subclasses.

A.3 SphericalParameterizedTriangleMesh

The `itk::SphericalParameterizedTriangleMesh` class is derived from `itk::ParameterizedTriangleMesh` and implements a surface parameterization for meshes of spherical topology (genus zero). Internally, the parameterization is represented as a second, spherical mesh with the

same topology as the original mesh. Coordinates of the spherical mesh are stored in a STL-array. Parameter coordinates are represented as 3D coordinates lying on the unit sphere. The mapping from parameter to object space is implemented by calculating the intersection of a ray (from the center of the sphere to a point indicated by the parameter coordinates) and the triangles of the sphere. If the intersecting triangle cannot be guessed, `MapCoordinates()` sorts all triangles according to their average distance to the parameter coordinates and starts intersection testing with the closest one. To map the parameterization to 2D-images, two patches are used (each mapping one half-sphere by means of the stereographic projection). New methods of the class are `GetSphericalMap()`, which returns a reference to the coordinate array of the spherical mesh and `InitializeSphericalMap()`, which reserves the necessary memory for the coordinate array and is usually called once by filters.

A.4 ConformalSphericalParameterizationFilter

The `itk::ConformalSphercialParameterizationFilter` class is derived from `itk::MeshToMeshFilter` and generates a conformal parameterization for meshes of spherical topology. It expects an `itk::IndexedTriangleMesh` as input and delivers an `itk::SphericalParameterizedTriangleMesh` as output. The implementation of this filter is based on the paper by Gu et al. [12]. Internally, the first step of the algorithm is to compute a Gauss map of the mesh, where all points are mapped to positions specified by their normal vectors. The protected method `ComputeGaussMap()` calculates all face normals coherently (i.e. pointing in the correct direction) and averages the face normals around each point. Subsequently, the string energy of the mesh is minimized in two steps, first optimizing the barycentric energy, then the conformal energy. In some rare cases, these optimizations do not converge and the filter will not return from the `Update()` function.

A.5 RotateSphericalParameterizationFilter

The `itk::RotateSphericalParameterizationFilter` is derived from `itk::MeshToMeshFilter` and rotates the parameterization of an `itk::SphericalParameterizedTriangleMesh`. A rotation of the parameterization results in rotating all mapped points/landmarks around the mesh. The function `SetTransform()` is used to specify the desired rotation by means of an `itk::AffineTransform`. Although this transform allows different transformations than just rotations, only rotations will result in a valid output.

A.6 GaussianWarpSphericalParameterizationFilter

The `itk::GaussianWarpSphericalParameterizationFilter` is derived from `itk::MeshToMeshFilter` and locally modifies the parameterization of an `itk::SphericalParameterizedTriangleMesh`. Within a local environment around a control point, all points of the spherical map (the parameterization) are moved in a specific direction. The magnitude of the movement is controlled by a Gaussian envelope function, its variance determines the area that is modified. The filter offers presets for different variances that can be activated by the `SetLevelOfDetail()` method. The number of available control points also depends on the level of detail (since fewer local environments fit on the sphere if each one is larger) and can be queried by `GetNumberOfControlPoints()`. When the level of detail is set, the control point that is used for the warp has to be specified using `SetActiveControlPoint()`. Subsequently, the direction and maximum magnitude of the movement has to be set using `SetDirection()`. The direction is specified by a three-dimensional vector: The first

element is the change in radians for the azimuth angle, the second is the change in radians for the polar angle. The third element is not used. Note that the filter only modifies the area around one control point at a time.

A.7 RemeshParameterizedMeshFilter

The `itk::RemeshParameterizedMeshFilter` is derived from `itk::MeshToMeshFilter` and maps a set of landmarks from parameter space to object space, effectively remeshing the input mesh. The method `SetLandmarks()` takes the landmark mesh in parameter coordinates. Each landmark point is mapped to object space using the `MapCoordinates()` method of the input mesh. If the input mesh has additional data associated with its vertices in the `PointDataContainer`, this data is also calculated for the new vertices (by interpolation within each triangle). The cell data for the output mesh is copied from the landmark mesh. Internally, the filter uses a cache to store all mapped vertices. If the parameterization changes around a certain point (invalidating the cached values), a local neighborhood is searched first to speed up the mapping process. These optimizations make the filter especially efficient if the used parameterization changes only locally, e.g. by using the `itk::GaussianWarpSphericalParameterizationFilter`.

A.8 ProcrustesAlign3DMeshFilter

The `itk::ProcrustesAlign3DMeshFilter` is derived from `itk::ProcessObject` and aligns a set of 3D-meshes or point sets in a common coordinate system using the generalized Procrustes matching [10]. All input meshes must have the same number of points with corresponding indices. The default behavior is to use a similarity transform to align all meshes to zero origin and scale them to best match a mean shape with norm one. The methods `AlignTranslationOn/Off()`, `AlignScaleOn/Off()` and `AlignRotationOn/Off()` can be used to deactivate certain classes of transforms. If the shapes should not be scaled to match norm one but retain their original sizes, set `UseScalingOff()`. Other options are `UseSingleIterationOn()` to run the algorithm only for a single iteration and `UseInitialAverageOn()` to start the optimization with an average of all input shapes instead of using a single one. Before calling any other methods, `SetNumberOfInputs()` should be used to specify the number of meshes to be aligned.

A.9 StatisticalShapeModel3DCalculator

The `itk::StatisticalShapeModel3DGenerator` is derived from `itk::Object` and finds the point correspondences across a set of two-manifold triangular meshes. After calling `SetNumberOfInputs()` to pass the number of meshes in the set, `SetInput()` can be used to specify the parameterized input meshes. Note that `SetInput()` copies the mesh information when the method is called, i.e. the mesh and its parameterization have to be initialized before.

In addition to the input meshes, the user has to specify a landmark mesh in parameter space that will be used to create the corresponding shapes. For the used spherical parameterizations, this means the desired number of vertices for the model should be spread equally on the unit sphere. The necessary coordinates can be obtained e.g. by subdividing one of the platonic solids and project all points to the sphere.

The search for optimal point correspondences is guided by a cost function that estimates the quality of a shape model. `SetCostFunction()` has to be called to specify the function to be used. The best results have been obtained with cost functions based on the Minimum Description Length of the shape model (as presented in [7]).

`SetOptimizeParameterizationStart()` determines if the algorithm should also optimize the start offsets of the parameterizations, which practically leads to optimizing the rotation of the sample meshes. If all samples have the same orientation, this can be turned off for a slightly faster optimization. The default is on. `SetUseAutomaticAlignment()` decides if a Procrustes alignment of the sample shapes and scaling to tangent size is conducted before building the model. The default `AutomaticAlignmentOn()` delivers the best results for the vast majority of cases.

A call of `Update()` triggers the optimization process, which can take several hours or even days to finish, depending on the hardware used and the number and complexity of sample meshes. The algorithm should scale linearly with the number of samples, the number of vertices on each sample and the number of landmarks. A rule of thumb is to decimate the sample meshes to reach a similar number of vertices as the used number of landmarks.

The optimization uses a gradient descent algorithm with fixed step size that can be set using `SetParameterizationWarpStepLength()` and `SetParameterizationStartStepLength()` for optimization of local correspondence and rotation, respectively. Convergence is determined by comparing the value of the cost function every 50 iterations. If the difference is less than the value set with `SetConvergence()`, the algorithm raises its level of detail. It stops when the maximum level is reached.

After that, the resulting point correspondences can be queried using three methods: `GetOutputMesh()` returns the corresponding points (landmarks) in the original coordinate system of each mesh, `GetOutputAlignedMesh()` returns the landmarks in the aligned coordinate system used to build the model and `GetOutputParameterization()` returns the optimal parameterization for each input.

A.10 StatisticalShapeModel3DGeneratorWithFeatures

This is a subclass derived from `itk::StatisticalShapeModel3DGenerator`. This class allows the usage of arbitrary features for the optimization of correspondence. The major change in functionality is in `InitializeMatrix()`, which uses the point data associated with the mesh, rather than vertex coordinates, for building the covariance matrix for MDL. Note that this class assumes that the input meshes have point data associated with their vertices. The point data should be of type `itk::Vector<double, n>` where *n* is the number of features being used. For example, for obtaining the results represented in Sect 5.2, the point data was of type `itk::Vector<double, 2>`, since there were 2 features per point (C and S).

A.11 ShapeModelCalculatorCostFunction

The `itk::ShapeModelCalculatorCostFunction` is derived from `itk::Object` and forms the base class for all cost functions to be used with the `itk::StatisticalShapeModel3DCalculator`. The cost function automatically maintains a pointer to the calculator that is using it. Since it is declared a friend of `itk::StatisticalShapeModel3DCalculator`, it can access all internal data of the model. Virtual functions that have to be implemented in subclasses are `GetValue()`, `GetGradient()` and `PrepareGradients()`. While `GetValue()` returns the current costs for the model, `GetGradient()` returns the gradient in a certain direction. How this direction is interpreted is specified by the implementation of `PrepareGradients()`.

A.12 VarianceBasedCostFunction

The `itk::VarianceBasedCostFunction` is derived from `itk::ShapeModelCalculatorCostFunction` and implements the `PrepareGradients()` method for all cost functions based on the variance, i.e. the eigenvalues of the model. Virtual functions that have to be implemented in subclasses are `GetValue()` and `GetGradient()`. Both can make use of the eigenvalue gradients stored in the `EVGradients` matrix.

A.13 SimplifiedMDLCostFunction

The `itk::SimplifiedMDLCostFunction` is derived from `itk::VarianceBasedCostFunction` and implements a simplified version of the MDL function originally presented by Davies [7]. Costs are calculated with the objective function of Thodberg [22]. Before using this cost function, the variance threshold that determines the amount of noise in the image data has to be set. The easiest way to set this threshold is to use `SetVarianceCutForModelRadius()` and pass the average voxel radius of the sample shapes as argument.

A.14 MeshASCIIReader

The `itk::MeshASCIIReader` is derived from `itk::MeshSource` and reads a mesh from disk. The mesh data is stored separately in two text files. The first file has a pts-extension and holds the vertices of the mesh: Each line represents one point by the x, y and z coordinates, separated by spaces. The second file has a fce-extension and stores the faces of the mesh: Each line represents one face by the indices of the involved vertices, separated by spaces. The first point in the pts-file has the index 0.

Currently, only triangular faces are supported by the reader. After specifying the file prefix (the name without the extension) with `SetFilePrefix()` and calling `Update()`, `GetReadError()` can be used to query the success of the operation. If no error occurred, `GetOutput()` returns the imported mesh.

If local features, rather than spatial locations of the vertices, are to be used for establishing correspondence, there should be a third file with the same name and a *.txt* extension. This file should have a header followed by the list of features for each vertex, indexed the same way as in the *.pts* file. The header should look like the following:

```
NUMBER_OF_POINTS = 4002
DIMENSION = 2
FEATURES = UNKNOWN
```

This means there are 4002 vertices in the mesh, and at each vertex, there are two features that should be used for the correspondence optimization. The 4002 lines following the header will consist of two values corresponding to the feature values at each vertex.

A.15 ParameterizedMeshASCIIReader

The `itk::ParameterizedMeshASCIIReader` is derived from `itk::MeshSource` and reads a spherical parameterized mesh from disk. The mesh data is stored separately in three text files. In addition to the pts- and fce-file that `itk::MeshASCIIReader` is using as well, there is a third file with a par-extension which stores the position of all points in parameter space. For spherical parameterized meshes, this position is saved as x, y and z coordinates of the point on the unit sphere, separated by spaces.

A.16 MeshFileWriter

The `itk::MeshFileWriter` is the base class for mesh writers in different formats. It is derived from `itk::ProcessObject` and offers a `Write()` method which calls `GenerateData()` to write the mesh to disk. In addition, there are several methods to get and set FilePrefix, Filename and FilePattern. The subclasses can decide which of these naming methods they want to use.

A.17 MeshSTLWriter

The `itk::MeshSTLWriter` is derived from `itk::MeshFileWriter` and stores a mesh in binary STL (stereolitography) format. Only triangular faces are stored. Use `SetFilePrefix()` to specify the `filename`, which will be extended with the stl-extension.

A.18 MeshASCIIWriter

The `itk::MeshASCIIWriter` is derived from `itk::MeshFileWriter` and stores a mesh in the ASCII format described in the documentation for the `itk::MeshASCIIReader` class. Currently, only triangular faces are stored. Use `SetFilePrefix()` to specify the `filename`, which will be extended with the pts- and fce-extensions.

A.19 ParameterizedMeshASCIIWriter

The `itk::ParameterizedMeshASCIIWriter` is derived from `itk::MeshFileWriter` and stores a spherical parameterized mesh in the ASCII format described in the documentation for the `itk::ParameterizedMeshASCIIReader` class. Use `SetFilePrefix()` to specify the `filename`, which will be extended with the pts-, fce- and par-extensions.

References

[1] J. Arvo. Fast random rotation matrices. In David Kirk, editor, *Graphics Gems III*, pages 117–120. Academic Press, 1992. 4.1

[2] C. Brechbühler, G. Gerig, and O. Kübler. Parametrization of closed surfaces for 3-D shape description. *Computer Vision and Image Understanding*, 61:154–170, March 1995. 3, 3.2

[3] A. D. Brett and C. J. Taylor. A method of automated landmark generation for automated 3D PDM construction. *Image and Vision Computing*, 18(9):739–748, 2000. 1

[4] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models – their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, 1995. 1, 2.1

[5] Rhodri H. Davies. *Learning Shape: Optimal Models for Analysing Shape Variability*. PhD thesis, University of Manchester, Manchester, UK, 2002. 4.3

[6] Rhodri H. Davies, Carole J. Twining, Tim F. Cootes, John C. Waterton, and Chris J. Taylor. A minimum description length approach to statistical shape modelling. *IEEE trans. Medical Imaging*, 21(5):525–537, 2002. 2.2

[7] Rhodri H. Davies, Carole J. Twining, Timothy F. Cootes, John C. Waterton, and Christopher J. Taylor. 3D statistical shape models using direct optimisation of description length. In *Proc. European Conference on Computer Vision, Part III*, pages 3–20. Springer, 2002. 1, 4.1, A.9, A.13

[8] A. Ericsson and K. Åström. Minimizing the description length using steepest descent. In *Proc. British Machine Vision Conference*, pages 93–102, 2003. 4.2

[9] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 157–186. Springer, Berlin, Heidelberg, 2005. 3

[10] C. Goodall. Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society*, 53(2):285–339, 1991. A.8

[11] Craig Gotsman, Xianfeng Gu, and Alla Sheffer. Fundamentals of spherical parameterization for 3D meshes. *ACM Trans. Graph.*, 22(3):358–363, 2003. 3.1

[12] Xianfeng Gu, Yalin Wang, Tony F. Chan, Paul M. Thompson, and Shing-Tung Yau. Genus zero surface conformal mapping and its application to brain surface mapping. In *Proc. IPMI*, pages 172–184, 2003. 3.1, A.4

[13] Tobias Heimann, Ivo Wolf, Tomos Williams, and Hans-Peter Meinzer. 3D active shape models using gradient descent optimization of description length. In *Proc. IPMI*, pages 566–577. Springer, 2005. 1, 5.1

[14] Dan Kalman. A singularly valuable decomposition: The SVD of a matrix. *College Math Journal*, 27(1):2–23, 1996. 2.1

[15] J. J. Koenderink. *Solid Shape*. MIT Press, 1990. 1, 2.3

[16] D. Meier and E. Fisher. Parameter space warping: Shape-based correspondence between morphologically different objects. *Trans. Med. Imag*, 21(1):31–47, 2002. 1

[17] Thomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997. 3.3

[18] Thodore Papadopoulo and Manolis I. A. Lourakis. Estimating the Jacobian of the singular value decomposition: Theory and applications. In *Proc. European Conference on Computer Vision*, pages 554–570. Springer, 2000. 4.2

[19] Rasmus R. Paulsen and Klaus B. Hilger. Shape modelling using markov random field restoration of point correspondences. In *Proc. IPMI*, pages 1–12, 2003. 1

[20] Christian R. Shelton. Morphable surface models. *Int. Journal of Computer Vision*, 38(1):75–91, 2000. 1

[21] Martin Styner, Kumar T. Rajamani, Lutz-Peter Nolte, Gabriel Zsemlye, Gábor Székely, Christopher J. Taylor, and Rhodri H. Davies. Evaluation of 3D correspondence methods for model building. In *Proc. IPMI*, pages 63–75, 2003. 1

[22] Hans Henrik Thodberg. Minimum description length shape and appearance models. In *Proc. IPMI*, pages 51–62, 2003. 2.2, 4.3, A.13

[23] Zheen Zhao and Eam Khwang Teoh. A novel framework for automated 3D PDM construction using deformable models. In *Proc. SPIE Medical Imaging*, volume 5747, 2005. 1