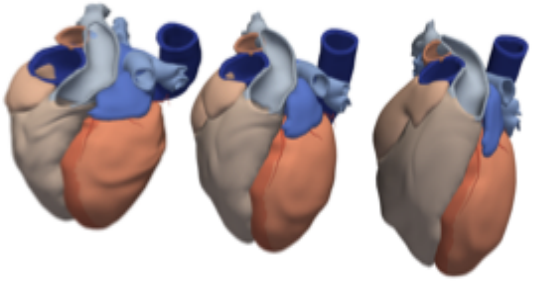
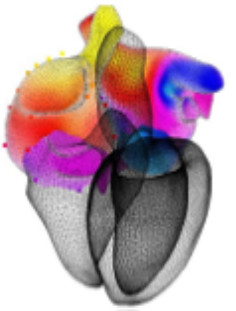
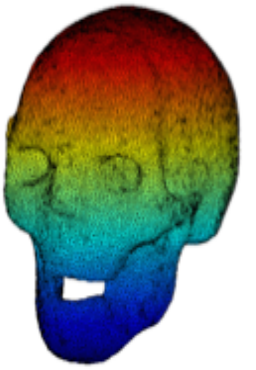
CISTIB Center for Computational Imaging & Simulation Technologies in Biomedicine

User guide: Sparse Statistical Shape Modeling v1.0

Nishant Ravikumar[1] and Ali Gooya[2]

[1] CISTIB Centre for Computational Imaging and Simulation Technologies in Biomedicine, Department of Mechanical Engineering, INSIGNEO Institute for *in silico* Medicine, The University of Sheffield, Western Bank, Sheffield, S10 2TN, South Yorkshire, United Kingdom

[2] CISTIB Centre for Computational Imaging and Simulation Technologies in Biomedicine, Department of Electronic and Electrical Engineering, INSIGNEO Institute for *in silico* Medicine, The University of Sheffield, Western Bank, Sheffield, S10 2TN, South Yorkshire, United Kingdom

Gooya, Ali, Christos Davatzikos, and Alejandro F. Frangi. "A bayesian approach to sparse model selection in statistical shape models" SIAM Journal on Imaging Sciences 8.2 (2015): 858-887.

The University Of Sheffield.

1 Dependencies

This is a guide for interested users/groups to download, compile and use the code developed at CISTIB (Center for Computational Imaging and Simulation Technologies in Biomedicine, The University of Sheffield, UK), to train 3D/4D statistical shape models using a sparsity-based approach (SpSSM). The code is written in c++ and has dependencies on the following open-source libraries:

- BLAS, GSL (download using sudo apt-get install or from Ubuntu software center)
- ALGLIB (Available at: http://www.alglib.net/download.php)
- ITK (Available at: http://www.itk.org/ITK/resources/software.html)
- Flann (Available at: http://www.cs.ubc.ca/research/flann/)

As of now the code can only be compiled on Linux-based OS due to dependency on BLAS-linear algebra and GNU scientific libraries and has been tested only on Ubuntu OS

The developed method can accommodate regular 3D point sets or 4D hybrid point sets where the 4^{th} dimension represents level set values in the narrowband around the boundary of a structure/ROI (in a segmented image). By default the code is set to work with 3D data. To work with 4D data, make the following changes to the **TypeDefinitions.h** header file located in the source folder prior to compilation:

- Change DataDimension=0 to DataDimension=1;

2 Download and Installation

The code is publicly available under the open-source GNU license at:
Third party libraries outlined above must be compiled prior to compiling the SpSSM code. CMake is required to configure ITK libraries and the developed SpSSM code. Following compilation of the dependencies, the following steps must be followed to compile and use the developed SpSSM code:

- Download and store SpSSM code in a 'source' folder and configure using CMake.
- Create new 'build' folder while configuring using CMake and switch to the corresponding directory in the terminal once the 'Makefile' has been generated ($CMake > Build > Configure > Generate$).
- Run 'make' to build executables.
- Edit .bashrc file and add the build directory address to the bottom of the file (export PATH:/.../.../). This allows the executables to be run from any folder. Add the directory address to the (DecimateMeshListUsingSparseModel.py) python script located in the source folder in the same manner. Now to run python scripts as executables, execute chmod +x DecimateMeshListUsingSparseModel.py in the terminal.

3 Usage Instructions

Once the code has been compiled and executables have been buit, basic usage instructions for each executable can be viewed by typing its name in the terminal (Ex: "sparseModel"). The pre-processing required involves generating segmentations (one or multiple-labels) of the structures of interest, which are inputs to the developed pipeline. The SSM training pipeline consists of the following steps (executables), outlined in the order they are to be employed:

1. Images to Point Sets: This is used to split multi-label segmentations into individual labels for individual structures and generate point sets from the latter. For example, given a multi-label segmentation of the brain where the cortex is assigned (Label=1) and the lateral ventricles (Label=2), this step can be used to generate point sets of both the cortex and ventricles individually. Multiple training samples can be processed automatically by maintaining a consistent pattern for file names (Ex: The multi-label segmentations for a training set should be named SegName01.nii, SegName02.nii, ..., SegName0n.nii).

   Usage: imagesToPointSets
   -n number of input images
   -f trainingFilePattern
   Options:
   -s startIndex (default: 1)
   -T levelSetThickness (default: 1.5mm, used only when 4D hybrid point sets are desired) -l lowerThreshold (default: .01)
   -u upperThreshold (default: inf)
   -o outputFilePattern (MESH%02d.txt)

   Note: The syntax '%02d' is used here and throughout to number the outputs in order, with their corresponding inputs

2. Decimate surface meshes: Due to the high memory requirement of fitting a Gaussian mixture model to a training set of points, it is recommended that the generated training surface meshes are decimated to < 10000 points each. A python script is provided **DecimateMeshListUsingSparseModel.py** which can be run as an executable (using chmod +x as described previously), to decimate the generated point sets. This executable employs ***sparsemodel*** to decimate training point sets. The outputs are decimated point sets written to ASCII .txt files.

   Usage: decimateMeshes
   -n number of input meshes
   -f trainingFilePattern
   Options:
   -s startIndex (default: 1)
   -r target reduction (default: 0.9, I.e. 90% percentage of decimation)
   -o outputFilePattern (MESHdec%02d.txt)

3. Run sparse model: This is the main component of the SpSSM training pipeline. The primary inputs to this executable are the decimated training point sets generated in the previous

step. A .txt file (Ex: InputMeshList.txt) containing the directory address for each training point set can be supplied as input rather than typing in the name of each decimated training point set. For example each line in the "InputMeshList.txt" file should contain: /home/Users/Documents/..../MESHdec%02d.txt

Usage: sparseModel
-n number of input meshes
-i input mesh list
-M GMM model file name. This can be an input or output file name (depending on option -m)
Options:
-N number of initial model points (default:1e4)
-c number of initial k-means pass (default:1)
-m set if estimation of model is desired (defualt:1)
-v set if virtual correspondance is desired (default: 1)
-s set if sparsity is desired (default:1)
-z desired sparsity amount (default:0.5, must between 0 to 1)
-t set if estimation of transformation is desired (default:1)
-r the maximum number of EM iterations (default: 100)
-g virtual aligned mesh generic file name (default: vir)
-T generic output transformations file name (default:tran)
-a model aligned mesh file generic name (default: alg)
-V virtual aligned mesh list file name (default: virList.txt)
-C a file name to keep all virtually correspondant points(default: virCorr.txt)
-A model aligned mesh list file name (default: algList.txt)
-L logfile name (default: logfile.txt)
-R variance reduction factor for classic model (default:0.85)

4. Compute PCA: Computes PCA over the virtual correspondence points stored in "virCorr.txt", output in the previous step following group-wise rigid registration of the training point sets using GMM and an EM-based framework. Outputs a .txt file containing the estimated eigenvectors and eigenvalues from singular value decomposition of the matrix of virtually correspondent points.

   Usage: computePCA
   -i inputAlignedXs.txt
   -o outputPCA.txt

5. Generate shape modes from trained SSM: Generate the mean shape and modes of variation in shape for the trained SSM. By default the extreme samples for the first three modes of variation are limited to $\pm 3\sqrt{\lambda_{i=1,2,3}}$ where λ_i represents the eigenvalue corresponding to a particular mode of variation (eigenvector).

   Usage: demoVariations
   -p pcaModelFileName (must be generated by "computePCA")
   -b bVectorsFileName
   Options:
   -o outputGenericFileName (default: var)

Here the "bVectorsFile" is a txt file specifying the desired modes of variation to be estimated. The format for the file is as follows: Text in red should be included in the file.
row 1: 0 0 0 −− > Computes mean shape
row 2: 1 0 0 −− > Computes mode 1, + stddev. shape
row 3: -1 0 0 −− > Computes mode 1, - stddev. shape
row 4: 0 1 0 −− > Computes mode 2, + stddev. shape
row 5: 0 -1 0 −− > Computes mode 2, - stddev. shape
row 6: 0 0 1 −− > Computes mode 3, + stddev. shape
row 7: 0 0 -1 −− > Computes mode 3, - stddev. shape

A sample "bVec.txt" file is located in the trunk/source folder.

6. Reconstruct surfaces from sparse point sets: A geodesic active contours (GAC) based framework is implemented to reconstruct surfaces from sparse point sets, which are output from the previous step (i.e. "demoVariations"). This step is necessary to generate surface meshes of the shapes drawn from the trained SSM space, useful for both visualization and subsequent application in physics-based simulations. This executable requires one text file (containing a point set/point coordinates) as input: "var%02d.txt", which is output from Step 6, whose surface is to be reconstructed. By default the mean shape is stored in "var001.txt" and can be recontructed in this manner.

   Usage: pointSetToSignedDistanceUsingGAC
   -i Input GMM text file generated from sparse model (containing sigma & priors) (Ex: GMM.txt)
   Options:
   -t templateImage (Ex: template.nii)
   -o outputImage (default: <inputPointSet>.nii)
   -x -y -z sampling distances (ignored if templateimage is provided, default: 1.0 1.0 1.0)
   -d additional proportional width used for framing (default: .2)
   -k curvature weight (default: 2.0)
   -g number of GAC PDE iterations (default: 1000)
   -T thickness of narrowband for svd made distance (default: 10)
   -N number of nearest neighbors (default: 20)
   -p propagation weight (default: 100, with neg in/ pos out)
   -c number of distance filtering iterations (default: 20)
   -m number of min flow (hole filling used for large convex structures) iterations (default: 0)
   -G set if intremediate images needed (default: 0)

   To summarise, this executable accepts a text file containing point co-ordinates as input and produces an implicit surface in the form of a signed distance map as output. Examples of the intermediary steps involved in this GAC-based surface reconstruction procedure are described by the figure below. Note: The images shown in the figure are examples of the images produced in the process, with the final step (7) representing the desired implicit surface of the input point cloud (in the narrowband). The signed distance map can subsequently be employed to generate a surface mesh by iso-surface extraction (functionality available on MATLAB for example) from the input point cloud.

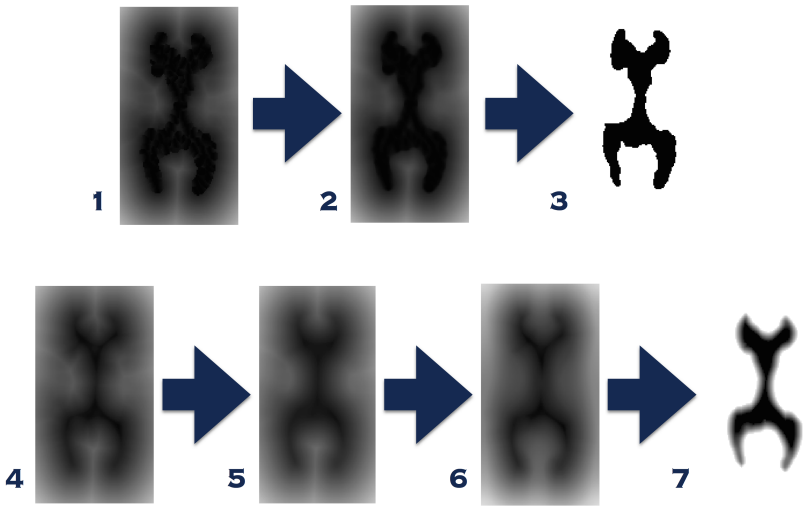


Figure 1: GAC-based implicit surface reconstruction: Intermediary images represent the steps involved in the pipeline. 1) Unsigned distance map, 2) smoothed distance map, 3) speed image, 4) hole-filled image (minimum curvature flow), 5) initial level set, 6) re-initialised level set image and 7) estimated signed distance map

7. Post-processing (MATLAB): Following SSM training and implicit surface reconstruction, the surface mesh representing any sample shape drawn from the trained SSM-space can be generated and visualised using the "niiImageToSurface.m" matlab function, example of reconstructed lateral ventricles shown below.Other matlab scripts and functions are also available to perform a variety of post-processing and SSM analyses such as: Computing the generalisation, compactness and specificity errors from cross validation experiments.

8. RBF-based surface approximation: The GAC-based implicit surface reconstruction procedure is time consuming. Once a dense surface mesh for the mean shape from the trained SSM is reconstructed the "warpPointSetUsingRBF" executable can be employed to reconstruct other mode shapes such that, they have the same number of points as the reconstructed mean surface mesh.

   Usage: warpPointSetUsingRBF
   -q queryPointSet text file including 3D point co-ordinates for reconstructed dense mean surface mesh
   -i inputPointSet text file including 3D point co-ordinates for original mean shape output from computePCA
   -f inputVectorField text file containing estimated eigenvalues and eigenvectors output from computePCA
   Options:

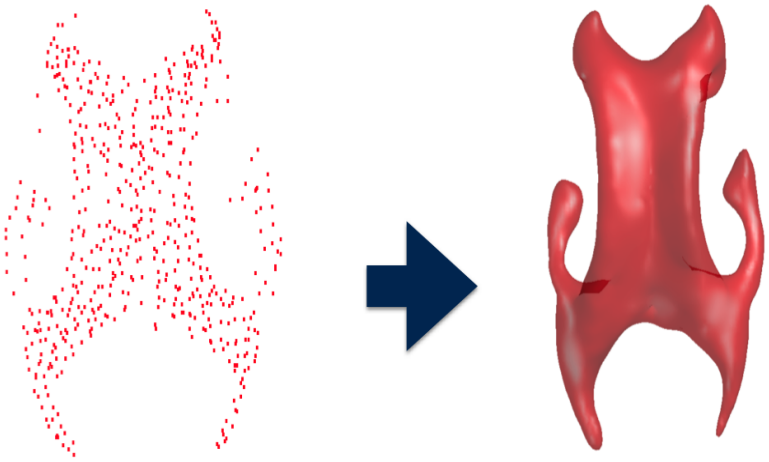


Figure 2: Reconstructed surface using pointSetToSignedDistanceUsingGAC to estimate implicit surface from a point cloud and niiImageToSurface.m to triangulate the zero-level isosurface from the estimated signed distance map in the narrowband around the surface/boundary

-w set if warping is needed (default: 0)
-o outputVectorField/PointSet (default: <inputVectorField>interp.txt/<queryPointSet>warped.txt)
-s scale for RBF interpolation (default: 4.0)

4 Compiling and usage issues

- It is crucial to ensure that the input training point sets to the pipeline are decimated to $\approx$ 10000 points each. The aim here is to maintain N_k i.e. total number of data points (N_n points * K shapes) to $< 200,000$ to $300,000$, due to the large memory requirements (>32GB RAM) . Otherwise, an error: "Segmentation fault (core dumped)" is thrown up in the terminal.