Project Report 2 : 3D Shape Search Engine

Brian R. Cairl

October 18, 2015

Introduction

This report details the design and testing of a 3D shape search engine which leverages several types of 3D shape descriptors. In particular, this search engine makes use of Geodesic distance (GD) and Euclidean distance (ED) shape descriptors (as well as a custom, curvature-base descriptor) to generate a pairwise similarity meaures (a *distance*) between 3D meshes. The contents of this report will include a description of the code written to generate both GD and ED descriptors, given a 3D shape (mesh) input; a description of how noise and distortion were added to a set of 3D models (TOSCA data set); a description of how the search-engine was implimented, and a corresponding GUI front-end for 3D shape retrieval; and an experimental comparison between GD and ED based searches when clean, distorted and incomplete models are presented to the aforementioned search engine. Additionally, a tutorial will be provided as to how to run the search-engine GUI.

In addition to aforementioned project requirements, this report will also present the result of a custom feature descriptor when applied to 3D shape matching. This custom descriptor, dubbed the *CD*, or curvature-dissimilarity descriptor, utilizes pair-wise dot products of unit normals which emenate from the triangular faces of each mesh. The formulation of this descriptor will be covered in detail at the end of this report, and will be compared to the performance of the GD and ED descriptors when applied to shape searching.

GD and ED Descriptors

Geodesic-distance (GD) and Euclidean Distance (ED) descriptors are histogram-based 3D descriptors used in the identification of 3D shapes. These descriptors are utilized to represent a 3D shape as high-dimensional feature vector. This section will describe the MATLAB implementation of a GD/ED descriptor generator used throughout this project.

GD and ED descriptors are generated using the MATLAB functions *create_GDdescriptor* and *create_EDdescriptor*, respectively. Each function takes

a single mesh-structure argument, by default. The mesh structure contains two member variables which contain a list of vertices, V, and edge connections, E, respectively. Within each of these functions, correspnding GD and ED values are calculated between some percentage of the vertices (to all other vertices) within each mesh and binned using a function *bin_values(d,dmin,dmax,dres)*, which creates a histogram (feature vector) from a vector of distances, d. Here we consider only a percentage of the mesh during feature-vector generation for the sake of saving computation time. The generated histogram has a minimum and maximum binning distance of $dmin$ and $dmax$; and a binning resolution of $dres$. Henceforth, distance values are counted into the jth bin of each ith histogram, H^i_j as follows:

$$H^i_j = \sum_{d \in D \text{ and } d < \text{dmax}} u(d) : u(d) = \begin{cases} 1, & (\text{dmin} + \text{dres} * j) < d < (\text{dmin} + \text{dres} * (j+1)) \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

To clarify, each histogram, H^i_j, is created using distances from the ith vertex to *all* other vertices in the provided mesh. Once histograms generated for each sample vertice, they are summed (in an element wise fashion) and then normalized by the number of sample vertexes considered in feature generation. We can, thus, wreite the final feature vector, $x \in \Re^N$, by:

$$x = \frac{1}{n_{\text{vectices}}} \sum_i H^i \tag{2}$$

The dimensionality of x, N, is determined by :

$$N = \text{ceil}\left(\frac{\text{dmax} - \text{dmin}}{\text{dres}}\right) \tag{3}$$

As for the distance calculations, seperate GD and ED functions have been written to handle the calculation of distance between a sample vertex and all other vertices in a mesh. To calculate the GD between points, a shortest path algorithm, $dijkstras(G, i)$ has been written which calculates theshortest distance between a node, i, to all other nodes in a graph G, as well as the associated pathing information. The graph G is represented as an adjacency matrix whose edge connections are computed as eucliden distances between

connected nodes. The graph G is generated from an input mesh M by a seperate function *mesh2graph(M,p)* which converts the mesh structure to the aforementioned adjaceny-matrix structure. The input parameter p is used to set the order of the L_p-norm distance to be computed between each graph node. To compute the euclidean distance between points, p is, obviously, set to $p = 2$.

To calculate the ED between points, an L_2 norm calculation performed on the vector difference between each ith sample vertex and all other vertices of the mesh M. This calculation is performed from within the driver function, *create_EDdescriptor*, aa a helper subroutine, *euclidean*.

Distance measure for shape matching

During shape-matching process, the distance (scalar dissimilarity) between shapes is determined by the distance between associated feature vectors. For a pair of meshes A and B with associated feature vectors x^A and x^B, shape dissimilarity, d_{AB} is calculated by:

$$d_{AB} = \sqrt{\sum_{i=0}^{N-1} (x_i^A - x_i^B)^2} \tag{4}$$

Noise and Mesh Distortion simulation

For the purpose of testing the robustness of the ED and GD descriptors, and in-turn the 3D shape search-engine, the provided (reduced) TOSCA dataset is augmented with noisey and incomplete versions of the original clean models.

Noisy Datasets

Noisy models are generated by adding Gaussian noise to each clean model in the original database. Each original mesh is modified using the provided function *addGaussianNoise* for three levels of Gaussian noise. Therefore, three (low, med, high) noisy data models are generated for each clean model. Noise levels are chosen to be 0.001, 0.008 and 0.015 for the low, medium and high noise models respectively. Noisy mesh generation is automated with a script named *noisy_query_set_generation*, which generates new noisy models from each original, clean model for each level of noise. Figure 1 shows a comparison between a clean and noisy model.

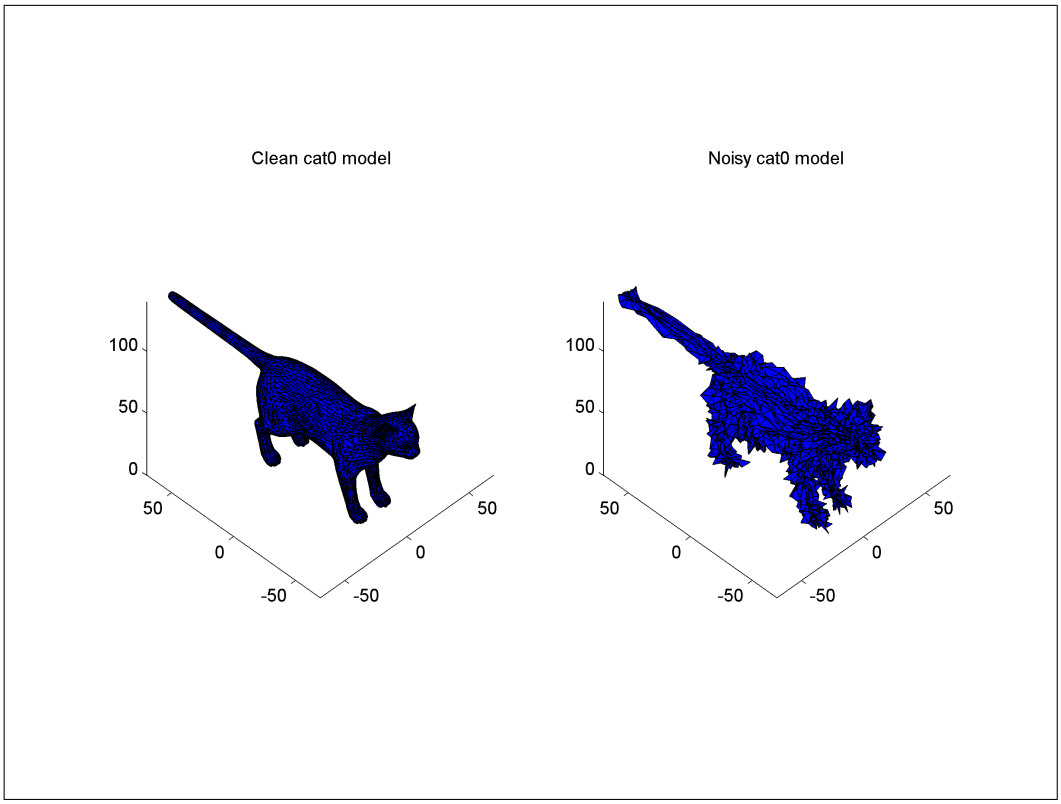

Figure 1: Comparison of a clean and noisy model

Incomplete Datasets

Three incomplete models are generated for each clean model by removing incrementally larger portions of each original mesh using the program MeshLab. Namely, low, medium, and highly effected models are generated by the aforementioned process. Any resulting holes in the effect mesh are filled using an hole-filling algorithm built into the MeshLab software. Figure 2 shows a comparison between a clean and incomplete model.

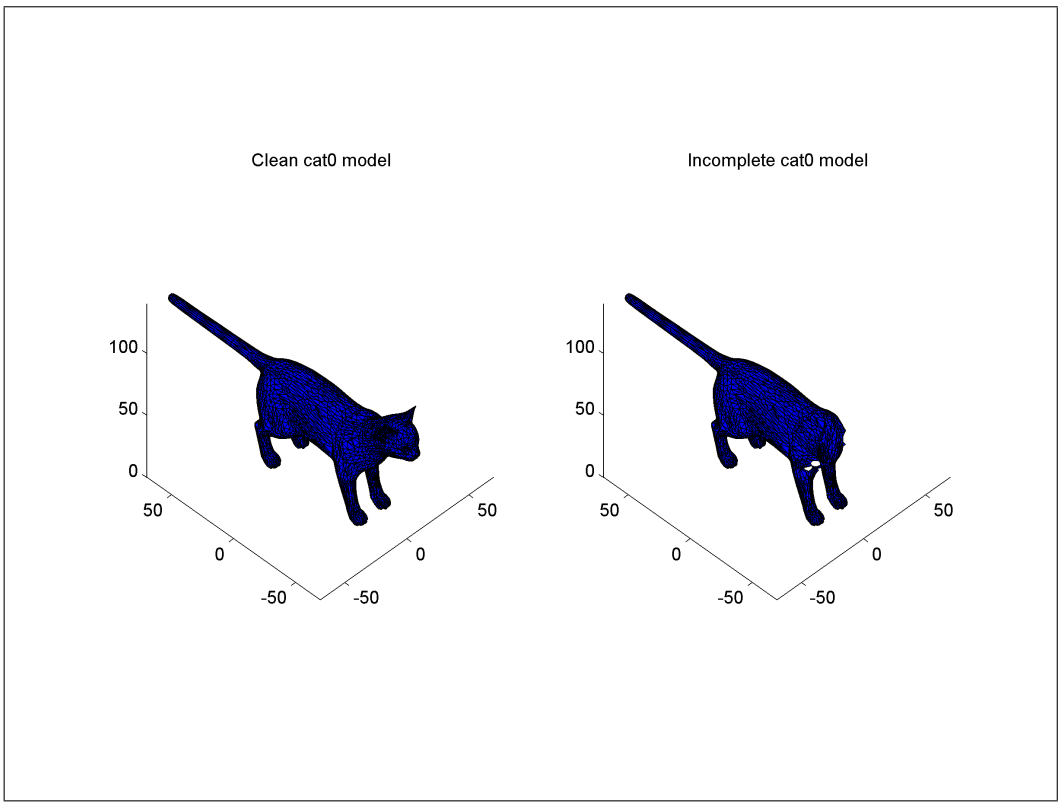

Figure 2: Comparison of a clean and noisy model

The 3D Shape Search Engine (and GUI)

How to run the Search Enginer GUI

Search Engine Results

A script, *search_engine_tester*, has been generated to automate perfomance tests for the generated search engine code.

Concluding Remarks