Technical Report No. 2013-610
Mesh Partitioning*

Bahram Kouhestani

School of Computing,
Queens University
Kingston, Ontario, Canada

kouhesta@cs.queensu.ca

September 16, 2013

*This technical report is based on the author's depth paper.

Contents

1 Introduction

Polygonal meshes are ubiquitous in applications where structure is applied to point data. For example in computer graphics solid objects are often represented by a mesh structuring sample points from an object surface. Finite element methods use meshes to provide structure for interpolating between discrete sample points.

A mesh is defined as a tuple of vertices, edges and faces. The process of generating a mesh, is to divide a given domain into small and simple pieces, called *elements*, with a few predefined properties. For example most applications require meshes which are well-shaped, non-uniform, conforming and respectful to the input shape. In order to achieve these properties, vertices of the mesh are not required to coincide with the point data and meshes are allowed to use additional points, called *Steiner points*.

Meshes can be categorized by their choice of elements. *Triangular meshes* are the most commonly used meshes in $2D$. Depending on the application and the desired quality, various different mesh generation approaches have been developed. These approaches can be categorized into three main classes: Advancing front methods [31], Delaunay refinement methods [9] and methods using a structured background [44]. Usually there is also a *clean up* component included in these algorithms that improves the quality of the resulting mesh. This improvement is done heuristically by finding a set of local transformations which replace a small set of triangles with other triangles of better quality [6].

Mesh partitioning (or *mesh segmentation*) has become a fundamental problem in many different computer graphics applications. Partitioning a mesh into smaller meshes benefits many different tasks including $3D$ printing, skeleton extraction, cognition, compression, texture mapping etc.

Given a $2D$ surface mesh (representing the surface of a $3D$ object), a mesh segmentation is defined as the set of sub-meshes induced by a partition of mesh elements into disjoint sub-sets. In addition, most applications impose constraints both on sub-meshes (such as connectivity) and on sub-partitions (such as a limit on the number of elements). Therefore, mesh segmentation can be viewed as an optimization problem of maximizing (or minimizing) a criterion function under a set of constraints.[35]. The choice of the criterion function heavily depends on the application in mind and different criterion functions such as planarity, curvature, geodesic distance have been employed. On the other hand, imposed constraints are generally either cardinality constraints, geometric constraints or topological constraints.

Mesh segmentation algorithms are categorized into *part type* or *surface type* segmentation. Part type (or shape-based) approaches partition the original object represented by a

mesh into meaningful, mostly volumetric parts. On the other hand, surface type (boundary-based) methods partition the surface mesh into patches under some criteria. It's worth mentioning that similar techniques are used to achieve these types of segmentation. Different automatic mesh segmentation algorithms have been developed in the past two decades.

Some of these algorithms include hierarchical mesh decomposition [2], decomposition using blowing bubbles [32], fuzzy clustering and cuts [19], watershed decomposition [29], Chopper framework [28] etc. Common techniques used in different algorithms are region growing, hierarchical clustering, iterative clustering, spectral analysis and implicit methods.

Comparing different mesh partitioning algorithms is another challenging issue. Visual properties of several mesh segmentation algorithms are compared in [1] by showing the resulting decompositions side by side. In [8] a benchmark for quantitative evaluation of mesh segmentation is introduced. This method uses human generated mesh decomposition and computes metrics to evaluate different automatically generated meshes.

This paper is organized as follows: Section 2 provides a quick overview of mesh generating algorithms. In section 3, we discuss mesh segmentation techniques and survey several segmentation algorithms. Section 4 discusses issues on comparing mesh segmentation algorithms and reviews one qualitative and one quantitative study on mesh segmentation. Finally, section 5 concludes the paper and looks at open problems and possible future directions in the field of automatic mesh segmentation.

2 An Overview of Mesh Generating Algorithms

We consider a company which builds electronic boards for a specific device. A board contains different components such as VLSI circuits, resistors, capacitors, etc. All of these components emit heat during operation. The board should be designed in a way that during the operation, its temperature always stays below a threshold. As the accumulation of the heat depends on the relative positions of components, the company should promote a process of simulating the heat emission of components. In order to carry out this task, a computer program simulates the heat emission of the board by utilizing finite element method. The first step of such a method is to build a mesh, which consists of dividing the board into small regions or elements. The regions are chosen small enough so that the heat emission within each element can be approximated and the mutual effect of neighbors can be simulated. The whole system is then solved numerically.

The above example is a simple illustration of the importance of mesh generating techniques and their application in the industry. Apart from their usage in finite element method, meshes are also extensively used in computer graphics and animations.
In this chapter, we review some of mesh generating techniques. We begin with a formal definition of a mesh.

A *mesh* is defined as a partitioning of a given geometric domain into small finite elements satisfying a number of constraints. The geometric domain is usually a set of points (a point cloud) or a polyhedron. The finite elements are usually simplices (triangles in two and tetrahedra in three dimensions) but can be more general shapes such as quadrilaterals or hexahedra. The constraints guarantee that the mesh contains "good" elements and may include many criteria. The quality of mesh elements highly depends on the application at hand. Later on we will discuss some of these constrains.

It is common to represent the surface of a 3D object with a 2D mesh. Such a mesh is called a *surface mesh*. In this paper we only consider surface meshes. A surface mesh M of a 3D object is represented as a tuple of $\{V, E, F\}$, Where $V = \{p_i \mid p_i \in \mathbb{R}^3, 1 \leq i \leq m\}$ is the set of vertices, $E \subseteq \{e_{ij} = (p_i, p_j) \mid p_i, p_j \in V, i \neq j\}$ is the set of edges and F is the set of faces. The most common meshes are triangular meshes where $F \subseteq \{f_{ijk} = (p_i, p_j, p_k) \mid p_i, p_j, p_k \in V, i \neq j, j \neq k, k \neq i\}$. Two faces in M are neighbors if they share a common edge.

The definition above gives the description of the geometry of a mesh (i.e the position and other geometric characteristics of each vertex). A more important aspect of a mesh is its topology. The topology of a mesh is explored using the concept of homeomorphism.

A *homeomorphism* is a bijective (one-to-one) continuous function $h : \mathbb{R}^n \to \mathbb{R}^m$ whose

inverse is also a continuous. Two sub-spaces are homeomorphic if there exist a homeomorphism between them.

Homeomorphism defines an equivalence relation among sub-spaces. Intuitively, two meshes are homeomorphic if their surfaces can be transformed to each other by using twisting, squeezing and stretching without glueing or cutting.

A mesh is a *k-manifold* or simply a manifold if every point on the mesh has a neighborhood homeomorphic to $\mathbb{R}^k$ or $\mathbb{H}^k$ (where $\mathbb{H}^k$ denotes the half-space of $\mathbb{R}^k$).

As a result in a 2-manifold mesh 1)each edge is adjacent to at most two faces and 2) faces incident to a vertex form a closed or open fan. Figure 1 illustrates the failure of the above conditions.

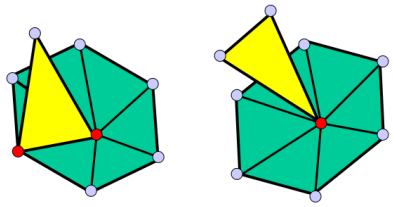

Figure 1: Example of two non manifold meshes.

The boundary of a manifold M is the set of points in M that has a neighborhood homeomorphic to $\mathbb{H}^k$. Note that if this set is empty then the mesh is without boundary.

In a surface mesh, boundary points are located on edges that are incident to only one face. These edges are called boundary edges. A *watertight* mesh, is a surface 2-manifold mesh without a boundary. Watertight meshes are essential for 3D printing.

Let M be a surface mesh and V' be a subset of the vertices of M. A *sub-mesh* M' is defined as the mesh with vertices V', edges $E' = \{(p_i, p_j) \in E \mid p_i, p_j \in V'\}$ and faces $F' = \{f_{ijk} = (p_i, p_j, p_k) \in F \mid p_i, p_j, p_k \in V'\}$.

The *dual graph* G of faces of M is constructed by representing each face in M with a node in G, and connecting two nodes in G if their corresponding faces share a common edge. Similarly, the dual graph of edges of M is constructed by representing each edge with a node and connecting nodes if their corresponding edges are neighbors.

The *orientation* of a face in a mesh is defined as an order of its adjacent vertices. The orientation of two neighbor faces is *compatible* if two vertices located on the common edge have opposite orders in faces. A manifold mesh is orientable if and only if each two adjacent faces have compatible orientations.

The most important aspect of a mesh is the quality of its elements. The following conditions are essential for almost all applications.
1. The mesh must be *conforming*: no vertex is allowed to reside on the interior of an edge.
2. The mesh must *respect* the geometric domain: The edges of the domain must be contained in the union of edges of the mesh.
3. The mesh must be *well-shaped*: The angles of no triangle should be too small or too big.
4. The mesh must be *non-uniform*: It should be fine near the boundary of the domain and coarse in the places away from the boundary.

In order to achieve these properties mesh generating algorithms produce new points in addition to points of the geometric domain. These new points are called *Steiner Points*. Depending on the application additional properties may be required. For example in mesh segmentation, the input mesh is usually assumed to be a manifold and watertight.

The next step after determining requirements is to generate a mesh with appropriate properties. In the following we will review some of the most common mesh generating techniques.

The history of mesh generating techniques can be divided into three periods [9]. Initially research in this field was done for using finite element methods. Techniques such as the Delaunay octree, advancing front methods, and mesh refinement techniques were developed during this period. These newly generated algorithms were good "enough" in practice but fragile and not robust. From around 1990 researchers in computational geometry became interested in the problem. Their goal was to find provably good mesh generating algorithms where results were guaranteed mathematically to produce satisfying meshes. By the year 2000, computer graphics used meshes extensively for many of their applications, most notably in computer animation. It is believed that nowadays, in economic terms, video game and motion picture industry exceed finite element industry in the usage of meshes [9].

Currently, there are three dominant classes of mesh generating algorithms; advancing front methods, Delaunay refinement algorithms and grid-based meshes.

Advancing front methods [31] start from the boundary of the domain and construct elements one by one going (advancing) inward. These methods produce high quality elements at the boundary, but the quality diminishes while advancing inward. Most advancing front

methods, first produce appropriated Steiner points and then triangulate the domain.

Delaunay refinement algorithms [9] first compute the Delaunay triangulation of the geometric domain and then refine it by adding Steiner points in a way that Delaunay properties of the mesh is preserved. Unlike front methods, Delaunay triangulation methods create their worst elements near the boundary of the domain. The main advantage of these methods is their mathematically guaranteed properties such as constructing a valid mesh and not producing skinny triangles.

In *grid-based methods* [44] the domain is overlaid by a grid. The grid is chosen small enough such that each grid cell covers only a simple and easily-triangulated portion of the domain. These methods produce excellent elements in the interior of the domain, but the quality of triangles near the boundary is worse than other methods. The disadvantage of grid meshes is that the elements are aligned in a few directions, their advantages are their speed, ease of parallelism, some mathematical guarantees and robustness to noisy input data. Also, graded meshes can be produced using variable-resolution grids.

Most mesh generation algorithms use a heuristic clean-up component to enhance the quality of elements. The mesh improvement is done by a set of local transformations, consisted of smoothing and topological transformation. The *smoothing* is applied iteratively to each vertex, improving the quality of its adjacent triangles. The simplest smoothing, called *Laplacian smoothing*, is to move the vertex to the centroid of vertices adjacent to it. Smoothing does not change the connectivity of the mesh. *Topological transformations* are operations that remove some elements from the mesh and replace them with a different elements in the same occupying region of the domain. The simplest topological transformation is the edge flip which replaces two triangles with two different ones with the same set of vertices (Figure 2). For surveys on mesh generation techniques see [9], [13] and [34].

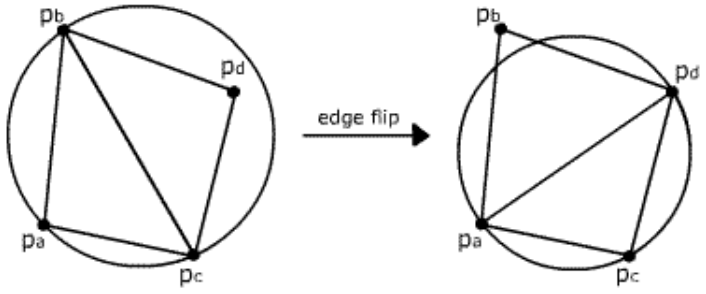

Figure 2: Edge flipping in a mesh.

3 Mesh Segmentation Algorithms

Automatic mesh segmentation (partitioning) has become an essential step in many geometric modelling and computer graphics applications. The extensive usage of mesh segmentation in diverse fields (such as 3D printing [28], object recognition [29], skeleton extraction [19], compression [17], simplification [11], parametrization [38], texture mapping [24], shape retrieval [48] and collision detection [25]) has attracted lots of attention to automatic mesh segmentation algorithms.
Some of techniques used in these algorithms have borrowed ideas from related fields such as image segmentation, finite element and simulation mesh segmentation, and clustering in statistics. Despite the similarity, techniques developed in these fields cannot be applied directly to geometric mesh segmentation. For example in finite element mesh segmentation, as the goal is to increase the load of each processor and to reduce the communication, the geometry of the shape is mostly neglected.

Different mesh segmentation algorithms seek distinct objectives; some try to partition the object which is represented by a mesh, while others try to extract patches with predefined properties. In this paper, we are going to review several algorithms of mesh segmentation.

Let M be a surface mesh and S be a set of elements (vertices, edges or faces), a *segmentation* Σ of M is the set of sub-meshes $\{m_1, m_2, ..., m_k\}$ induced by a partition of S into k disjoint sub-sets.

Note that if S is the set of vertices or the set of edges then some faces of M are not included in Σ, these are the faces with vertices in two different sub-sets of S. Segmentation algorithms which partition on V or E use a post-process to join these faces to their appropriate neighbors.

The objective of a mesh segmentation is to generate a partitioning with some predefined properties and constraints on sub-sets of S or sub-meshes of Σ. Therefore, we can define mesh segmentation as an optimization problem [35]:

Given a surface mesh M and a set of elements S, find a disjoint partitioning of S into $S_1, S_2, ..., S_k$ such that the criterion function $J : 2^S \rightarrow \mathbb{R}$ is maximized or minimized under a set of constraints C.

The criterion function and constraints are defined according to the application in hand.

The set of constraints puts limitations on sub-sets or induced sub-meshes or both. For example, we may seek a partitioning into convex sub-meshes where J -the number of parts- is minimized. This problem is studied in [7] and is shown to be NP-complete, which suggests that most reasonable mesh segmentation problems tend to be hard.

Therefore, algorithms resort to approximation solutions in order to achieve a feasible computation time. These approximation solutions commonly use one of the following approaches:
1. Region growing,
2. Hierarchical clustering,
3. Iterative clustering,
4. Spectral analysis and
5. Implicit methods.

Later in this chapter we will review an algorithm for each approach.

Another distinction between mesh segmentation algorithms is their view of the given geometric object; a 3D volumetric view or a 2D surface view. According to this, two types of segmentation are defined, *part type* segmentation and *surface type* segmentation.
In a part type segmentation, the goal is to partition the object, defined by a mesh, into meaningful or semantic components, and in general to create volumetric parts. This type of segmentation is appropriate for 3D printing, skeleton extraction and feature extraction. In the surface type, surface patches are created using the geometry of the mesh. This type of segmentation is often used in texture mapping, compression and simplification.

Depending on the application, different constraints are applied to sub-sets and sub-meshes of a segmentation. Possible constraints are of three major types:
1. Cardinality Constraints: Constraints on the sub-sets of S such as a bound on the maximum/minimum number of elements to eliminate small or large partitions, a bound on the ratio between the maximum and minimum number of elements in all parts to create a balanced partition, and a bound on the maximum/minimum number of segmentations ($|S|$).
2. Geometric Constraints: These constraints are imposed on the geometry of sub-meshes. Some common geometric constraints are maximum/minimum area of sub-meshes, maximum/minimum length of the diameter or perimeter of sub-meshes, and convexity of patches.
3. Topological constraints: These constraints are applied to the shape of sub-meshes; restricting each sub-mesh to be topologically equivalent to a disk and/or restricting each sub-mesh to be a single connected component.

The factor that has the most influence on a segmentation is the criteria of selecting

elements to be in a single sub-mesh. These criteria are used to produce sub-meshes with particular attributes. Different attributes such as planarity [11], curvature [22], geodesic distance [27], symmetry [28], and convexity [7] are used in different studies. A criterion uses a metric (usually L_2 or L_∞-metric) to evaluate an attribute of a sub-mesh and then the sub-mesh is selected if the evaluation is above a threshold. It is also common for algorithms to use multiple criteria and attributes [28, 11, 37].

Next, we review mesh segmentation techniques and examine an example of each technique.

3.1 Single Source Region Growing

The simplest technique in mesh segmentation is *single source region growing*. Region growing is a local greedy approach which starts a sub-mesh cluster from a seed element and gradually grows the cluster by adding new elements. Elements are added to the cluster as long as they satisfy particular criteria. Then a new seed is chosen and the process continues until all faces belong to a cluster.

As in [35], this technique can be summarized as follows:

```
Initialize a priority queue Q of elements
Loop until all elements are clustered
    Choose a seed element and insert into Q
    Create a cluster C from seed
    Loop until Q is empty
        Get the next element s from Q
        if s can be clustered into C
            Cluster s into C
            Insert s neighbors to Q
Merge small clusters into neighboring ones
```

Region growing algorithms differ from each other according to criteria that determine if an element can be added to the cluster. It is common to choose the priority of elements as their order in a depth first search (DFS) or breadth first search (BFS) traversal and to select seeds randomly. Also most algorithms utilize a post processing component for smoothing or straightening borders of resulted regions and to merge small sub-meshes into one of their neighbors.

As an example of a region growing method, we are going to review the flooding algorithm for convex decomposition of a polyhedral surface [48, 7]. Note that a patch is convex if it

lies completely on the surface of its convex hull.

Let P be a polyhedral surface represented by a mesh M with n vertices. The goal is to partition M into k disjoint convex patches $m_1, m_2, ..., m_k$ whose union is M by using the minimum number of patches.

In [7], it is shown that this problem is a special case of clique partitioning, therefore, the problem is in NP. It can also be shown that convex partitioning is NP-complete by a reduction from Satisfiability Problem (SAT) [7]. Thus, algorithms mainly aim to construct a convex partitioning of the given domain.

One way of achieving this goal is to use an incremental heuristic algorithm called *flooding heuristic*. The flooding algorithm starts with an empty cluster and a random node (seed) in the dual graph of the given mesh. It traverses the dual graph from the seed and collects faces as long as the resulting patch remains convex. Then a new random seed is chosen and the process repeats until all faces belong to a patch. The traversal method is either a DFS or BFS which in practice result in similar numbers of patches.
Convexity can fail in two ways, global failure or local failure. A local failure occurs when an edge at which a face is attached to the patch is concave. In other words, in local failure the dihedral angle of the edge (the angle between faces adjacent to the edge) is less than π. In a global failure although the patch is locally convex at all edges, some faces fail to reside on the boundary of the convex hull. In practice global failures occur rarely, but they are the only reason why the surface convex decomposition problem is hard [7]. By ignoring the global failures, all greedy algorithms -regardless of the order in which they choose triangles- will result in the optimal solution. Also without global failure, covering and partitioning are the same, and allowing convex patches to overlap will not change the number of patches.

The major drawback of the flooding algorithm is the number of patches it produces. For some specific examples this can be off the optimal by a factor proportional to n. As a result in [7], *flood-and-retract* algorithm is introduced. In the first phase of flood-and-retract algorithm, flooding is used to produce convex patches which may overlap. Next overlapped patches are retracted until each face belongs to only one patch. This is done by keeping a queue of faces to be retracted. First, for each patch, all the faces along the boundary that also belong to another patch are inserted in the queue. Then the face at the top of the queue is retrieved and it is removed from its patch if the patch remains connected. After removal, adjacent faces (in the same patch) that belong to multiple patches are inserted randomly to the queue. This process is done iteratively until all patches are retracted and the queue becomes empty. Then, the resulting cover is transformed into a partitioning in an arbitrary manner.

The advantage of flooding is its simplicity and ease of implementation. Also the flooding algorithm can be used with other criteria and is not limited to convexity.

Another issue in convex segmentation is the geometry of the given object. For large "real-world" models, regardless of the algorithm, small concavities will result in a large number of patches. Small concavities are due to local failures by very small angles, and may result in patches with few faces. Two solutions are proposed in [48] to cope with this problem. First, dihedral edge angles less or equal to $\pi + \varepsilon$ (for a small predefined value of ε) are considered to be convex. Although in this way patches are not necessarily convex, the resulted segmentation is more suitable for meaningful decompositions.
Second, a post-processing step is added where small patches (area-wise) are merged. Here, patches are sorted increasingly according to their surface area. Starting from the smallest, every small patch is merged with its largest neighbor until the area of all patches are greater than $q * A$, where q is a user defined value and A is the total area of the given model.

3.2 Multiple Source Region Grow

The single region growing technique can be improved by using multiple seeds and advancing in parallel. The generic algorithm for multiple source region grow is as follows [35]:

```
Initialize a priority queue Q of pairs
Choose a set of seed elements {si}
Create a cluster Ci from each seed si
Insert the pairs <si, Ci> to Q
Loop until Q is empty
    Get the next pair <sk, Ck> from Q
    if sk is not clustered already and sk can be clustered into Ck
        Cluster sk into Ck
        For all un-clustered neighbors si of sk
            insert <si, Ck> to Q
Merge small clusters into neighboring ones
```

The major drawback of multiple region growing is its dependence to the initial selection of seeds where sometime random seeds result in a "bad" segmentation. In the watershed growing technique (defined below), this is solved by starting at height function minima.

The *watershed growing algorithm* [48], originally developed for image segmentation, is a well-known multiple source algorithm where seeds for growing regions are found based on a height function.
Let $h(x, y) : S \to \mathbb{R}$ be a height function. A *catchment basin* is the set of points whose path

of steepest descent ends in the same local minimum of h. The idea of the algorithm is to select minima as seeds and grow regions using catchment basin. It proceeds as follows. First, all local minima are computed and labelled. Then, flat areas, which are either plateaus or minima, are found. If the flat area is a minimum then it is treated like one and a label is given to it. For a plateau, steepest gradient descent is used to loop through it until a labelled region (or point) is found. The label of the plateaus is then set with the same label of the found region and it is joined to this region. Similarly, the remaining unlabelled vertices are descended until they are joined to labelled regions. The resulted regions comprise the final decomposition.

This technique utilizes the idea of following the path of a water-drop downward on a hill. Similar to single growing, this method may produce a large number of small segmentations. Therefore, a post-process merges adjacent regions whose watershed depths are below a certain threshold.

The remaining issue is the choice of the height function. In image processing, this is done by choosing the height to be the grey-level of a pixel. For mesh segmentation the height function is defined on the edges of the polyhedral mesh. Let the height function $h = 1 - \cos(\alpha)$, where α is the dihedral angle of the edge. Then the watershed algorithm is applied to edges rather than vertices of the mesh. For the final segmentation, each face is associated with its lowest height edge.

An example of such segmentation is shown in Figure 3(a), where resulted regions are illustrated with different colors. Note that flat regions are minima with the height function value of 0.

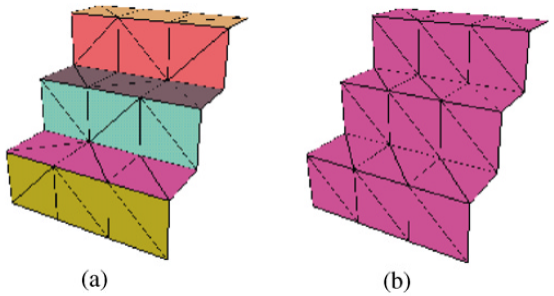

Figure 3: Watershed decomposition [48].

The above definition of the height function has the advantage that the topology of the mesh has no effect on the final segmentation. As a result, the mesh in Figure 3(b) will be parti-

tioned to the exact same regions.

The watershed growing algorithms can be found in many variations, where the difference is mainly the definition of the height function. For instance in [47] the average geodesic distance (AGD) of vertices of the mesh is used for the height function. In [43] the mesh is considered as a conduct object with electrical charge, where the charge density at each point defines the height function.

3.3 Hierarchical Clustering

Region growing algorithms are "local-greedy" and sometimes result in unsatisfactory global-solutions. In contrast, hierarchical clustering algorithms are "global-greedy", as they tend to merge cluster pairs with the minimum merge cost among all possible merge operations. Initially, in a hierarchical clustering, each face is considered as a separate cluster and then in each step two adjacent clusters, with the minimum merging cost, are joined together. The generic algorithms is as follows [35]:

```
Initialize a priority queue Q of pairs
Insert all valid element pairs to Q
Loop until Q is empty
    Get the next pair (u,v) from Q
    if (u,v) can be merged
        Merge (u,v) into w
        Insert all valid pairs of w to Q
```

Hierarchical clustering algorithms mainly differ from each other according to merging criteria. Next we review the hierarchical mesh segmentation algorithm presented in [11].

Given a 2-manifold mesh M representing the surface of a (complex) 3D object, the goal is to find a hierarchical partitioning of faces of M into clusters in order to ease the processing of the object.

This problem is tackled by introducing a hierarchical clustering where clusters represent aggregate properties of the original surface at different scales. The dual graph of the clusters of the mesh is used to achieve the hierarchy. In this graph each node corresponds to a face cluster, which is a connected set of faces. Initially each cluster contains only one face. These clusters form leaves of the hierarchy. In each step an edge of the dual graph is selected and contracted. An edge contraction merges two adjacent nodes of the graph into one which corresponds to grouping two neighbor clusters into a single one.

The edge selection is done using a simple greedy algorithm. Each dual edge is assigned with a "cost" and in each iteration the edge with the minimum cost is contracted. At the

top of the hierarchy, when the algorithms runs to its fullest, a single cluster represents the whole surface of the object. Note that although all clusters are connected, they are not necessarily simple and may contain holes.

Iterative dual edge contraction results in a natural hierarchy. Whenever two vertices are merged, both become the children of their new parent node and the parent node corresponds to the union of its children's associated clusters.

The cost of each edge contraction depends on criteria with which the quality of clusters is evaluated. The primary criterion used in [11] is the planarity of clusters, which measures how well a cluster can be approximated by a plane. Suppose a cluster is represented by a plane (n, d) with the normal n and offset d. The distance of a point v (represented by a vector) to this plane is $n^T v + d$. Now the *fit error* of the given plane $n^T v + d = 0$ for a cluster is the average squared distances of all vertices of the cluster to the plane:

$$E_{fit} = \frac{1}{k} \sum_{i=1}^{k} (n^T v_i + d)^2$$

The least squared best plane is the one that minimizes this E_{fit}. In order to compute this plane efficiently a *fit quadric* is defined:

$$P_i = (A_i, b_i, c_i) = (v_i {v_i}^T, v_i, 1)$$

$$P_i(n, d) = n^T A_i n + 2{b_i}^T (dn) + c_i d^2$$

Using this fit quadric the inner term of E_{fit} can be written as

$$(n^T v_i + d)^2 = (n^T v_i + d)({v_i}^T n + d) = n^T (v_i {v_i}^T) n + 2dn^T v_i + d^2$$

Now the error is evaluated using a corresponding set of quadrics:

$$E_{fit} = \frac{1}{k} \sum_i P_i(n, d) = \frac{1}{k} \left(\sum_i P_i \right) (n, d)$$

Where the addition of quadrics is defined by component-wise addition of matrices, vectors and scalars.

Every dual node in the hierarchy will be associated with a fit quadric P and a best-fit plane (n, d), where $P(n, d)$ measures the planarity of the cluster. Now the cost of an edge contraction is defined as the sum of fit quadrics of its endpoints, $(P_i + P_j)(n, d)$.
This novel representation of the metric is the main contribution of the paper and is essential for the overall efficiency of the algorithm. As we will see later it bounds the time complexity

of the algorithm to $O(nlogn)$.
Next issue after merging two clusters is to find the optimal plane which minimizes $P(n, d)$. Based on principal component analysis (PCA) the sample covariance matrix is used:

$$Z = \frac{1}{k-1} \sum_{i=1}^{k} (v_i - \bar{v})(v_i - \bar{v})^T$$

where $\bar{v} = \left(\sum_i v_i\right)/k$. The eigenvector of Z with the smallest eigenvalue is the normal of the least squared best plane.
In the case where Z has equal eigenvalues, for example when vertices of the cluster reside on a sphere, the best fit plane may not be uniquely defined and any of the possible orientations can be chosen.

As we are only concerned about the eigenvectors of Z, we drop the $1/(k-1)$ averaging factor from Z, and rewrite Z in terms of a fit quadric:

$$Z = \sum v_i {v_i}^T - k(\bar{v}\bar{v}^T) = A - \frac{bb^T}{c}$$

So, the optimal fit plane is also computed using the fit quadric, where the normal of this plane, n, is the eigenvector of $A - bb^T/c$ with the smallest eigenvalue, and as the plane passed through the mean, the offset $d = -n^T\bar{v} = -n^T b/c$.

The planarity criterion alone is not enough for many applications. For objects with local fold-backs, although the error fit is low, the normal of the cluster may not be a good fit for all cluster faces. Thus an additional error term, which measures the average deviation of normals, is used:

$$E_{dir} = \frac{1}{w} \sum_i w_i (1 - n^T n_i)^2$$

where w_i is the area of the face f_i and $w = \sum_i w_i$. This metric can be also expressed as a quadric:

$$E_{dir} = \frac{1}{w} \sum_i w_i R_i(n) = \frac{1}{w} \left(\sum_i w_i R_i \right)(n)$$

Where

$$R_i = (D_i, e_i, f_i) = (n_i {n_i}^T, -n_i, 1)$$
$$R_i(n) = n^T D_i n + 2{e_i}^T n + f_i$$

Using both these metrics, the algorithm begins by setting the quadrics P and R for each face of the mesh. Then the cost of each initial edge contraction is set as the sum of the quadrics of its endpoints and the optimal plane representing both faces (with the error $E_{fit} + E_{dir}$) is

computed. After the initialization, dual edges are placed in a min-heap, keyed on the cost of the contraction, and in each step the edge with the lowest cost is contracted. The quadrics of the new node is set as $P_i + P_j$ and $R_i + R_j$ and the costs of edges connected to this new node are updated. The algorithm repeats until the heap is empty.

Figure 4 shows an example of the above segmentation.

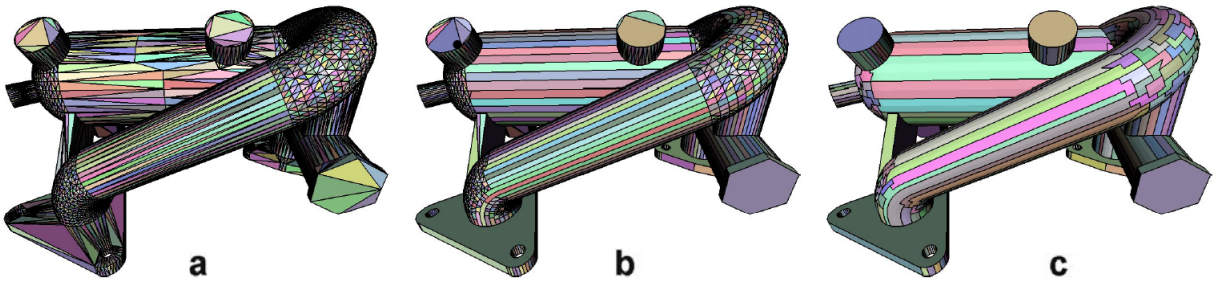

Figure 4: Hierarchical face cluster (a) Original mesh with 11,036 faces (b) Segmentation with 6000 clusters (c) Segmentation with 1000 clusters [11].

As Figure 4 illustrates, when planarity and orientation are the only criteria, the algorithm may produce long skinny patches. These skinny patches are problematic in many applications such as simplification or radiosity simulation. In order to solve this problem a shape regularity metric is introduced.

Given a cluster with the area w and perimeter p, the irregularity of the cluster, denoted by γ, is defined as

$$\gamma = \frac{p^2}{4\pi w}$$

The irregularity can be seen as the ratio of the squared perimeter of the cluster to the squared perimeter of a circle with the same area as the cluster. Therefore, the irregularity of a circle is 1 and the irregularity of a skinny long shape is a larger value. Then E_{shape} metric for an edge contraction is defined as

$$E_{shape} = \frac{\gamma - \max(\gamma_1, \gamma_2)}{\gamma}$$

where γ_1 and γ_2 are the irregularity of the two merged clusters and γ is the irregularity of the resulted cluster.

The total error metric used in the algorithm combines all three previous metrics:

$$E = E_{fit} + \alpha_1 E_{dir} + \alpha_2 Eshape$$

where α_1 and α_2 are two user-defined constants. For results reported in the paper, α_1 and α_2 are considered to be 1. Further examination of potential values of these two constants are left as a prospective study.

The algorithm has an overall time complexity of $O(nlogn)$: In the initialization phase, quadrics are computed in $O(n)$ time, this is due to the planarity of initial clusters where each cluster consists of only one face. In the clustering phase $O(n)$ operations are performed on the heap, where each operation takes $O(logn)$ time. Therefore, the overall time complexity is $O(nlogn)$. Note that the computation of the shape metric is done by recoding the perimeter of each cluster and saving the length of the boundary where two clusters share. Therefore, the computation of this metric does not enforce any overhead to the overall complexity.

3.4 Iterative Clustering

In previous methods, the number of clusters can not be predicted in advance. Thus, they are sometimes called as non-parametric techniques. In some cases the exact number of clusters is given a priori. Then the optimal segmentation can be sought iteratively. This search method, which is called parametric search, follows the basis of the K-mean algorithm. The process begins with k representatives and finds k clusters using these representatives. New representatives are calculated for each cluster and the process repeats until no representative changes or after a certain number of iterations.

The generic algorithms is as follows [35]:

```
Initialize k representatives of k clusters
Loop until representatives do not change
    For each elements s
        Find the best representative i for s
        Assign s to the ith cluster
    For each cluster i
         Compute a new representative
```

The clusters are built with the idea that elements inside a particular cluster should be closest to the representative of that cluster compared to all other representatives. However, it is common for iterative algorithms to use a region grow method to create clusters from calculated representatives. The reason lies in the fact that a proper distance metric is hard

to find, for example, the Euclidean distance is not appropriate for manifolds and geodesic distance is very expensive to compute. Therefore, iterative methods resort to region growing algorithms to construct clusters.

In the following we review an iterative segmentation [37] developed particularly for the application of morphing polyhedral surfaces.

Given a surface mesh M with n vertices and a value k, the goal is to partition M into k meaningful disjoint patches.

As usual, the most important issue is to decide which faces should be clustered together. Here, the assumption is that two distant faces, both in terms of physical distance and angular distance, are less likely to end up in the same patch. Therefore, the distance metric is defined as follows:

Let F_1 and F_2 be two adjacent faces of the mesh with the dihedral angle of α. The distance between F_1 and F_2 is

$$Distance(F_1, F_2) = (1 - \delta)(1 - \cos^2(\alpha)) + \delta Phys\_Dist(F_1, F_2)$$

where Phys_Dist is the sum of the distances between centres of mass of the two faces and the midpoint of their common edge, and δ is a real number between 0 and 1 which controls the relative importance of angular distance and physical distance. Note that the first part of the distance measures the angular distance and the second part measures the physical distance of two faces. This particular definition of Phys_Dist makes it independent to the dihedral angle.

If F_1 and F_2 are not adjacent, then:

$$Distance(F_1, F_2) = \min_{F_3 \neq F_1, F_2} (Distance(F_1, F_3) + Distance(F_2, F_3))$$

The approach of the algorithm is to iteratively enhance segments by locally switching faces to improve a global function. The algorithm consists of four steps:

1. Preprocessing: In the first step distances between all adjacent faces are computed.

2. Selecting the initial representatives: Although the initial representatives can be chosen randomly, as the initial selection can affect the result of the segmentation and the number of iterations, they are chosen as follows: The first representative is chosen as the face with the minimum distance between its center of mass and the center of mass of the mesh. Then, using Dijkstra's algorithm, the minimum distance of all faces to the representative is computed.

Iteratively, as long as the number of selected representatives is less than k, a new representative is selected as the face with the maximum average distance to all existing representatives .

3. Computing the segmentation: For each face, its distances to all representatives are calculated. Each face is assigned to the patch with the closest representative. Note that this process results in connected patches, but patches may not be simple and can have holes.

4. Re-electing the representatives: The ultimate goal of the algorithms is to select representatives such that the function

$$F = \sum_{p} \sum_{f \in patch(p)} Dist_{fp}$$

where $Dist_{fp}$ is the distance between the face f and the representative p of its patch, is minimized. This can be done by selection the representative of each patch as the face which optimizes the function $\min_p \sum_f Dist_{fp}$. The other option is to simply choose the face whose centre of mass is closest to the centre of mass of the patch. The latter has a much better complexity and in practice seems to produce better results.

The major benefit of the algorithm is that the number of patches is controlled by the user and can be set small enough to avoid over-segmentation. This is particularly useful in morphing as only patches which represent meaningful components are useful. In the initial step, the system can also determine a suitable number of patches by adding a new representative as long as its distance to any existing representative is larger than a threshold, where the threshold is a predefined value times the diameter of the object.

3.5 Spectral Analysis

In spectral graph theory the Laplacian matrix of a graph G is used to extract different characteristics of G. The Laplacian matrix of G is defined as the matrix $L = D - A$, where A is the adjacency matrix of G and D is a diagonal matrix with the element d_{ii} as the degree of vertex i.

Using the first k leading eigenvectors of L (with the highest eigenvalues), G is embedded into the space $\mathbb{R}^k$ and the graph partitioning problem is reduced to a geometric space partitioning problem [16]. Spectral analysis can be also used for mesh segmentation. This technique was first utilized in [27] as follows.

Let M be a given manifold with n vertices. The algorithm aims to partition M along concave seams into k partitions, where k is the number of desired segments.

The first step is to defined an appropriate distance metric between faces of M, which concurs with the aim of the algorithm. Let F_1 and F_2 be two adjacent faces with the dihedral angle of α. The angular distance between F_1 and F_2 is defined as

$$Ang\_Dist(F_1, F_2) = \eta(1 - \cos(\alpha))$$

Let $Geod(F_1, F_2)$ be the geodesic distance between the centres of mass of F_1 and F_2, and $avg(Geod)$ be the average geodesic distance between all the adjacent faces. Similarly, let $avg(Ang\_Dist)$ be the average angular distance between all adjacent faces. The distance between F_1 and F_2 is defined as

$$Dist(F_1, F_2) = \delta \cdot \frac{Geod(F_1, F_2)}{avg(Geod)} + (1 - \delta) \cdot \frac{Ang\_Dist(\alpha)}{avg(Ang\_Dist)}$$

Then the dual graph of M is constructed and the weight of each dual edge is set as the distance between the corresponding (adjacent) faces in M. The distance between two arbitrary faces is defined as the weight of the shortest path between their dual nodes on the dual graph. In the preprocessing step, distances between all faces are calculated. The value δ is set close to zero, e.g. $\delta \in [0.01, 0.05]$ to put more emphasis on the angular distance. The value of η is chosen from $[0.1, 0.2]$.

To this end we have calculated a weighted graph where weights between closer nodes are smaller than others. In order to extract an adjacency-like matrix, where elements with higher values denote closer nodes, the affinity matrix is introduced. The *affinity matrix* encodes the likelihood that two faces cluster together and is defined as the symmetric matrix $W \in \mathbb{R}^{n \times n}$, where

$$W(i, j) = e^{-Dist(F_i, F_j)/2\sigma^2}$$

and $\sigma = \frac{1}{n^2} \sum_{(1 \leq i,j \leq n)} Dist(F_i, F_j)$.

Note that in the definition above, $0 \leq W(F_i, F_j) \leq 1$, and closer faces have a larger affinity. In practice, the above value of σ seems to work fine, but other choices are also possible.

The dual graph of M and the affinity matrix W comprise the input of the spectral analysis. Let matrix D be a diagonal matrix with the element d_{ii} as the sum of the i-th row of W. D is used to normalized W as following

$$N = D^{-1/2} W D^{-1/2}$$

The resulting matrix N is a symmetric matrix with elements $N_{ij} = W_{ij}\sqrt{D_{ii}D_{jj}}$. Let $e_1, e_2, ..., e_n$ be eigenvectors of N corresponding to eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$. The

rows of matrix $V = [e_1, e_2, ..., e_k]$ define an embedding of the original data points into a k-dimensional space. Normalizing rows of V will further simplify this embedding. Let matrix $\hat{V}$ be the matrix constructed by normalizing rows of V. This matrix is an embedding of N on a unit sphere in k-dimension.

According to the *Polarization Theorem* [5] embedded points of high affinity (points with a higher possibility of being clustered together) will move towards each other on this sphere, while other pairs move apart. This implies that the segmentation of the points on this sphere is easier than the original points and a simple clustering algorithm such as K-mean clustering (which uses Euclidean distance) will solve the problem.

In order to initialize the K-mean algorithm the association matrix $\hat{Q} = \hat{V}\hat{V}^T$ is used. From linear algebra, it is known that the $n \times n$ matrix $\hat{Q}$ is the best rank-k approximation of N [10]. The smallest element of $\hat{Q}$, $\hat{Q}_{rs}$, designates the farthest apart pair of points on the embedded space. The elements r and s can be used as the first two seeds of the K-mean clustering, and then other seed are selected to minimize the maximum association among the existing seeds. This process advantages the algorithm in two aspects, first it gives a good initialization of the K-mean clustering and second it can be used to find a proper number of clustering. When adding a new seed dramatically increases the maximum affinity then no more seeds are added.

It is worth to note that although embedding in a lower k-dimension will result in an easier clustering problem, small value of k does not benefit the segmentation. The reason is, by using smaller number of eigenvectors the distortion associated with the embedding coordination increases. The rule of thumb is to use the same number of eigenvectors as the number of desired clusters.

Additional notes: Authors claim that one of the advantages of the algorithm is its efficiency, by stating that computing a few leading eigenvectors is quite efficient using ARPACK. However, as the construction of the affinity matrix requires $O(n^2)$ time and space, this algorithm is unusable for complicated models. The execution time of the algorithm on an Intel Xeon 2.8 GHz machine with 1GB RAM is reported to be 29.57 seconds for a model with 400 faces which further indicates that the algorithm is not as efficient as it is claimed to be. The main advantage of the algorithm may be its ease of implementation, especially when an existing K-mean clustering algorithm is used.

3.6 Implicit Methods

Some algorithms construct the segmentation implicitly by defining the boundaries between sub-meshes rather that partitioning the set of vertices or faces [23]. Others may utilize dif-

ferent structures of the mesh, such as the skeleton, to produce the desired segmentation [25]. Also, similar to the idea of using multiple criteria, some algorithms may combine different methods in the hope of avoiding drawbacks of a particular method and to benefit from the strength of others [26].

More recent studies have considered new directions for mesh segmentation. For instance in [18] a hierarchical pose invariant mesh segmentation is proposed. This segmentation is particularly useful to partition dynamic and flexible objects. Next we review this algorithm.

Given an orientable mesh M, the goal is to hierarchically partition M into meaningful components. Also the resulting segmentation must be invariant both to the pose of the model and to different proportions between the model's components.

The algorithm constructs a hierarchy tree, where each node is associated with a sub-mesh. The root of the tree is associated with M. For each node, the following steps are performed (see below for the explanation of each step):
1. *Mesh coarsening*: The original mesh is simplified by retaining some particular vertices of the mesh and deleting the rest using a simple mesh coarsening algorithm. Mesh coarsening benefits the algorithm in two ways. It accelerates the algorithm by reducing the complexity and also it decreases the sensitivity of the algorithm to the noise.
2. *Pose invariant representation*: Using *Multi-dimensional scaling*, the mesh is transformed to a canonical mesh where Euclidean distances between vertices in the canonical mesh are similar to the geodesic distances between their corresponding vertices in the original mesh.
3. *Feature point detection*: A few points, called prominent feature points, are computed on the transformed mesh and their corresponding vertices are identified. Feature points have the property of being invariant to the pose of the object.
4. *Core extraction*: The core component is extracted using a spherical mirror.
5. *Mesh Segmentation*: Using feature points and the computed core, the algorithm computes other segments.
6. *Cut refinement*: The boundaries between segments are refined to go along the natural creases of the mesh.
7. *Mesh refinement*: The segmentation (on the coarse mesh) is mapped to the original mesh and the cut refinement is applied once again.

As mentioned earlier Multi-dimensional scaling (MSD) is used to transform meshes into pose invariant representations. MSD uses the dissimilarity information of data and represents dissimilarities as Euclidean distances in a k-dimensional space. Note that it is not necessary for MSD to preserve the ratio of dissimilarities. Here, the dissimilarity is defined as the geodesic distance between two vertices. This is due to the property that the geodesic distance is invariant to poses and bending has a small effect on it. Thus, the dissimilarity

matrix is defined as $\{\delta_{ii} = GeodDist(v_i, v_j)\}, 1 \leq i, j \leq n$.

Let d_{ij} be the Euclidean distance between the corresponding vertices of v_i and v_j in the transformed space. The MDS algorithm [20] iteratively attempts to optimize the following stress function

$$F_s = \frac{\sum_{i<j}(f(\delta_{ij}) - d_{ij})^2}{\sum_{i<j} d_{ij}^2}$$

where f is an optimal monotonic function of dissimilarities: First, the algorithm computes an initial configuration of points in the k-dimensional MDS space, denoted by S_{MSD}. This can be done by a random sampling. After calculating distances d_{ij} between all points in the MDS space, the algorithm finds the optimal monotonic function (using pool adjacent violators [3]). Then points of the original mesh are re-mapped to the MDS space resulting in the next set of points. This process repeats until the stress function is sufficiently small. For the use of segmentation a 3-dimension MDS space is used (i.e. $k = 3$). Figure 5(a) illiterates the MDS representation of the dino model.

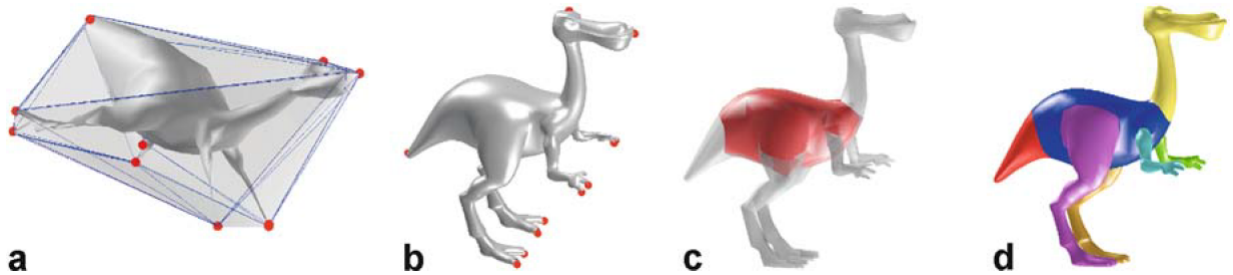

Figure 5: (a) Feature points in MDS representation (b) Feature points on the original model (c) The core (d) Final segmentation after refinement [18].

The MSD representation is also used for defining prominent feature points. Feature points are both located on the tip of a prominent component and are pose-invariant. The first condition is a local condition and is expressed as follows: Let N_v be the set of neighbor vertices of a vertex v. v resides on the tip of a component if it is a local maximum of the sum of the geodesic distance function. In other words, $\forall v_n \in N_v$:

$$\sum_{v_i \in S} GeodDist(v, v_i) > \sum_{v_i \in S} GeodDist(v_n, v_i) \quad (1)$$

In the MSD presentation, tip points are extreme in some direction. Using this fact, a prominent feature point is defined as a mesh vertex which its MDS representative resides

on the convex-hull of the MSD representation and satisfies the Eq. 1. The computation of feature points is now straight forward, first the convex hull of S_{MSD} is computed, and then all vertices on the convex hull which satisfy Eq. 1 are reported. (Figures 5(a) and 5(b)).

In contrast to feature points, vertices on the core tend to reside near the centre of the MSD representation. A spherical mirror is used to reverse this situation, so that vertices of the core become external and easy to extract by simply computing the convex hull the projection points. Let C be the centre of mass of the set of vertices in S_{MSD} and let $R = max_{v \in S_{MSD}} \|v - C\|$. Obviously a sphere with the centre C and radius R will be a bounding sphere for the set of points of S_{MSD}. Considering this sphere as a mirror, each vertex $v \in S_{MSD}$ is projected to

$$v' = v + 2(R - \|v - C\|)\frac{v - C}{\|v - C\|}$$

Intuitively, the projection of v lies on the ray connecting the centre of the mirror to v and have the same distance as v to the point of the mirror which lies on this ray. Figure 6 depicts the spherical mirroring in 2D and 3D.

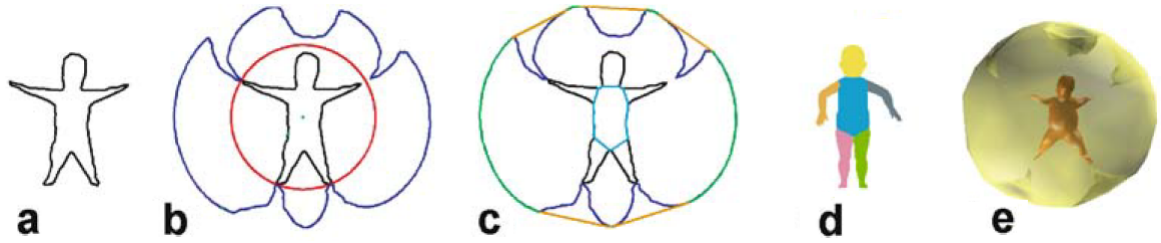

Figure 6: Spherical mirroring, (a) MDS representation of the object (b) The mirror and projected points (c) The convex hull of projected points (d) The final segmentation (e) The 3D view of the spherical mirroring [18].

In order to compute the core of the object the convex hull of mirrored vertices is computed. The vertices that reside on the convex hull, along with their faces, comprise the initial core component. If the resulting core separates all the feature points then this component is the actual core of the object. Otherwise some feature points lie on the same component. In this case the core is extended iteratively by adding neighbor faces. This is done until all feature points are separated or the last extension reduces the distance from the core to the closest feature point by more that a constant factor (0.5). In the latter case the algorithm backtracks to the state where the last separation of a feature point occurred and the core extraction process is finished. Note that possibly un-separated feature points are separated

at a finer level of the hierarchy.

When the core is computed the segmentation is done by removing the core and considering each connected component with at least one feature point as a new node in the hierarchy. Components without a feature point are joined to the core.

Computed segments of the previous step may have coarse boundaries. Therefore, in the last step of the algorithm boundaries are refined. The idea is to produce better located boundaries which pass through short concave edges. This causes the cut to look more natural. Refinement is performed by using a flow graph constructed on a search region. The search region is defined as the set of all faces whose shortest distance to the boundary is smaller than a factor of their distance to the nearest feature point. As an example in Figure 6(e) the search region of the boundary between the left leg and the torso may contain the left thigh and the lower parts of the torso. The flow graph is constructed by adding two new vertices S and T to the dual graph of the search region. As the search region contains two boundaries, S is connected to all the corresponding nodes of one of these boundaries and T is similarly connected to the other one.

Weights of edges of the flow graph are set using a capacity function. Different capacities are used in different studies. The capacity function can be defined in a way that the capacity of shorter and concave edges is higher. As the minimum cut of the flow graph tends to pass through arcs with small capacities, it defines the appropriate boundary.

4 Comparing Mesh Segmentation Algorithms

In the previous section we reviewed several segmentation techniques and discussed their metrics and selection criteria. The immediate question that comes to mind is which algorithm works the best, both in general and for a specific application. However, answering this question is rather a difficult task for several reasons. First, different algorithms do not attempt to detect same features of the input object. Also, the class of input objects varies for different applications and consists of smooth featureless objects to very complex ones. Second it is not easy to formally define the features of a shape which in return makes it very difficult to perform a quantitative analysis of segmentation algorithms. Finally, segmentation algorithms are completely application-based and are devised to solve a specific problem which further complicates their comparison.

In the literature on mesh segmentation, there are very few studies addressing this problem. In [1], five part-type segmentation algorithms are reviewed and their performances are compared by showing results of segmentations side by side. Recently a benchmark for

quantitative evaluation of mesh segmentation algorithms is presented in [8]. Concurrently in [4], with a similar methodology, a framework for quantitative evaluation is presented. The benchmark of [8] includes more human subjects, models and evaluation metrics. In this chapter we review [1, 8].

The following algorithms are compared in [1]:
1. *Mesh decomposition using fuzzy clustering and cuts* [19]: An iterative algorithm which aims to find meaningful components using clustering and refines the boundaries with graph cuts.
2. *Mesh segmentation using feature points and core extraction*[18]: The implicit algorithm explained in section 3.6.
3. *Tailor multi-scale mesh analysis using blowing bubbles* [32]: A multiple source region growing algorithm which uses a set of spheres placed on vertices of the mesh.
4. *Plumber mesh segmentation into tubular parts* [33]: Another multi source region growing algorithm which decomposes the shape into tubular features.
5. *Hierarchical mesh segmentation based on fitting primitives* [2]: A hierarchical algorithm which uses a set of pre-defined primitives (planes, spheres and cylinders) to approximate meaningful components of the shape.

Models used for segmentation consist of medical models, CAD models, human figures, animal figures and miscellanea class. Figure 7 illustrates performances of these algorithms on 4 models from the animal category.

By evaluating the qualitative performances of algorithms, the paper concludes that
1) Algorithms which consider concave features may be more suitable to segment natural models, while algorithms which extract geometric properties are usually more appropriate for CAD applications.
2) Defining precise geometric metrics is easier for CAD models and methods like Plumber which are based on a priori knowledge of the feature, perform better on CAD models.
3) Algorithms which consider curvature in their metrics are more sensitive to the pose of the model. MDS can be applied as a preprocessing in order to make algorithms less sensitive to poses.
4) There is no perfect segmentation algorithm for all classes of models. Each algorithm has its benefits and drawbacks.

A hidden goal in most mesh segmentation algorithms is to imitate human visual perception and produce segments similar to the ones created by humans. This suggests the idea of gathering a database of human made segmentations and compare results of segmentation algorithms with this ground-truth. This idea is leveraged in [8] to construct a benchmark

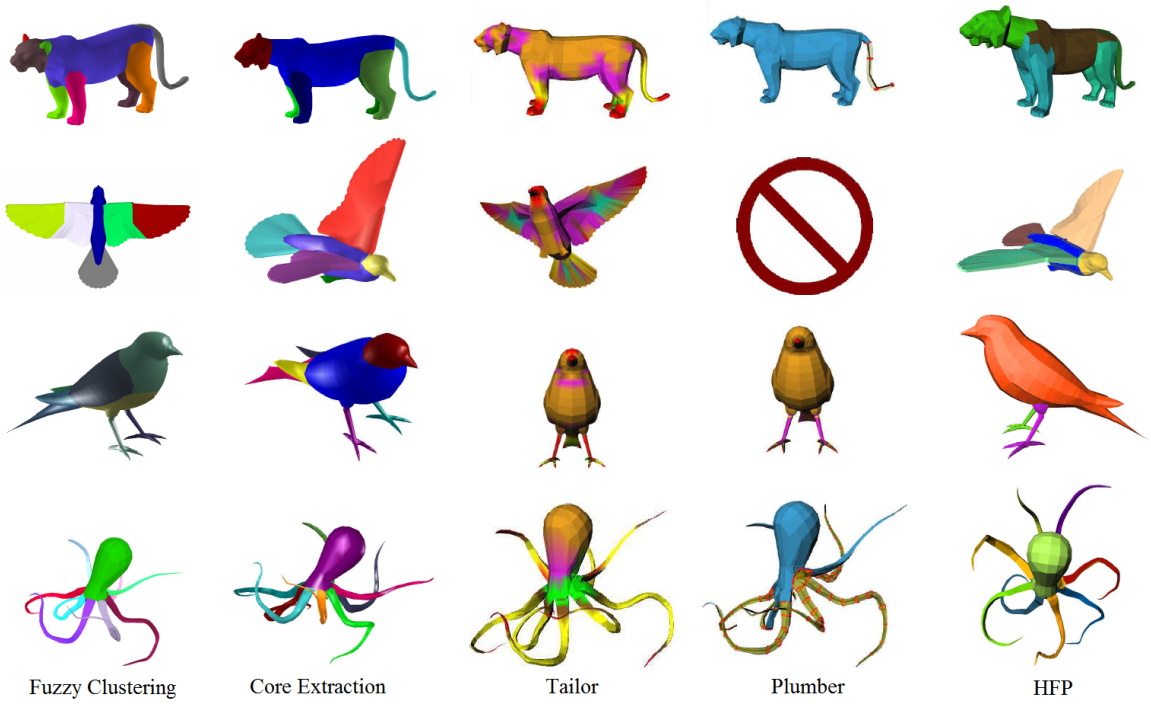

Figure 7: Qualitative comparison between five segmentation algorithms [1].

for quantitative evaluating of 3D mesh segmentations.

In order to construct this benchmark, authors of [8] hired 80 people to manually segment 380 surface meshes of 19 different categories, which resulted in 4,300 human generated segmentations with an average of 11 segmentations per model. Models are represented by 2-manifold watertight meshes and are chosen from 2007 SHREC Shaped-based Retrieval Contest. Categories include human bodies, chairs, mechanical CAD parts, hands, fish, pliers, etc. Meshes were partitioned using an interactive tool which allows users to select points along cuts (segment boundaries) by clicking, and the system connects points through their shortest connecting path. Users can also adjust current cuts by inserting or deleting points along the cut. People were recruited through Amazon's Mechanical Turk (www.mturk.com), and meshes were randomly distributed among them. Participants were asked to segment models into "functional parts". During the period of one month, 4,365 segmentations were received from which 365 were rejected for not having any cuts. 25 were over-segmented and 353 had at least one cut that seemed to be an outlier. However, these segmentations were accepted to avoid bias in the results.

After gathering the database, the next step is to develop metrics to compare human

generated segmentations with computer generated ones. Four metrics are developed where one measures how close segment boundaries of two segmentations are (boundary-based) and other three measure the consistency of segment interiors (region-based). These measure are as follows:

1. Cut Discrepancy: This metric measures the distance between the closest cuts of the ground-truth and the generated segmentation. Let C_1 and C_2 be the sets of all vertices on the segment boundaries of segmentations S_1 and S_2, respectively. The geodesic distance between a vertex $p_1 \in C_1$ to the set of cuts C_2 is defined as

$$Geod\_Dist(p_1, C_2) = \min\{Geod\_Dist(p_1, p_2), \forall p_2 \in C_2\}$$

The Directional Cut Discrepancy, DCD($S_1 \Rightarrow S_2$), of S_1 with respect to S_2 is defined as

$$DCD(S_1 \Rightarrow S_2) = mean\{Geod\_Dist(p_1, C_2), \forall p_1 \in C_1\}$$

Cut Discrepancy, CD(S_1, S_2) is defined as the sum of the directional functions in both directions divided by the average Euclidean distance from a vertex of the mesh to the centroid of the mesh (denoted by $avgRadius$):

$$CD(S_1, S_2) = \frac{DCD(S_1 \Rightarrow S_2) + DCD(S_2 \Rightarrow S_1)}{avgRadius}$$

The above definition ensures symmetry of the metric and avoids scale effects.

2. Hamming Distance: Hamming Distance is used to measure the overall region-based difference between two segmentations. Let $S_1 = \{S_1^1, S_1^2, ..., S_1^m\}$ and $S_2 = \{S_2^1, S_2^2, ..., S_2^n\}$ be two segmentations with m and n segments, respectively. The Directional Hamming Distance is defined as

$$D_H(S_1 \Rightarrow S_2) = \sum_i \left\| S_2^i \backslash S_1^{i_t} \right\|$$

Where "\" is the set difference operator, $\|x\|$ is the total area of faces in x and $i_t = \arg\max_k \left\| S_2^i \cap S_1^k \right\|$. This metric finds the best correspondence in S_1 for each segment in S_2 and sums up the set differences. If S_2 is regarded as the ground truth then missing rate R_m and false alarm rate R_f is defined as follows:

$$R_m(S_1, S_2) = \frac{D_H(S_1 \Rightarrow S_2)}{\|S\|}$$

$$R_f(S_1, S_2) = \frac{D_H(S_2 \Rightarrow S_1)}{\|S\|}$$

The Hamming Distance is then the average of these two rates:

$$HD(S_1, S_2) = \frac{1}{2}(R_m(S_1, S_2) + R_f(S_1, S_2))$$

Note that this metric is sensitive to differences in granularity, but when correspondences are correct, it gives a more meaningful evaluation.

3. Rand Index: This metric measures the likelihood that a pair of faces are either in the same segment in two segmentations, or in different ones in both. Let f_i^1 and f_i^2 be the segment IDs of face i in S_1 and S_2 and n be the number of faces in the original mesh. Rand Index is defined as

$$RI(S_1, S_2) = \binom{n}{2}^{-1} \sum_{i,j,i<j} [C_{ij}P_{ij} + (1 - C_{ij})(1 - P_{ij})]$$

Where $C_{ij} = 1$ iff $s_i^1 = s_j^1$ and $P_{ij} = 1$ iff $s_i^2 = s_j^2$.

Note that when $CijP_{ij} = 1$, face i and j have the same id in both segmentations and when $(1 - Cij)(1 - P_{ij}) = 1$ face i and j have different IDs. So $RI(S_1, S_2)$ measures the proportion of face pairs that agree or disagree on their segment ID in S_1 and S_2. In order to make this metric consistent to previous ones, $1 - RI(S_1, S_2)$ (which denotes dissimilarities) is reported.

4. Consistency Error: This metric measures the hierarchical similarities/dissimilarities in segmentations. Let $R(S, f_i)$ denote the sub-segment which contains the face f_i. The local refinement error is

$$E(S_1, S_2, f_i) = \frac{\|R(S_1, f_i) \backslash R(S_2, f_i)\|}{\|R(S_1, f_i)\|}$$

If n is the number of faces, then Global Consistency Error (GCE) and Local Consistency Error (LCE) are defined as

$$GCE(S_1, S_2) = \frac{1}{n} \min\{\sum_i E(S_1, S_2, f_i), \sum_i E(S_2, S_1, f_i)\}$$

$$LEC(S_1, S_2) = \frac{1}{n} \sum_i \min\{E(S_1, S_2, f_i), \sum_i E(S_2, S_1, f_i)\}$$

Note that GCE forces all local refinements to be in the same direction (i.e it considers one of the segmentations to be strictly higher in the hierarchy), while LCE allows refinement in both directions. Thus, $GCE(S_1, S_2) \geq LCE(S_1, S_2)$. Both of these metrics are symmetric and measure the nested, hierarchical differences in two different segmentations. Their main

disadvantage is that they give unreal scores when one of the segmentations is hugely over-segmented or under-segmented. For example the error is zero if one of the segmentation has only one sub-set.

The last issue before comparing segmentation algorithms is to set parameters of each algorithm. As we have seen in the previous section, most segmentation algorithms require the number of segmentations as a given value. This is solved by simply averaging the number of segments in human-made segmentations of a particular model and forward this value to segmentation algorithms. Although this solution gives a disadvantage to completely automatic segmentation algorithms, its rationality is that a typical user has a good estimation of the number of segments.

In order to test the benchmark authors compare 7 segmentation algorithms (K-mean [37] explained in section 3.4, random walks [21], fitting primitives [2], normalized cuts [15], randomized cuts [15], core extraction [18], and shape diameter function [36]). The comparison is done in different levels by comparing a segmentation to the average human-made segmentations. Also, each human segmentation is compared to others by holding it out and comparing it to other segmentations of the same model in the benchmark. The results indicate that humans are indeed very consistent with each other than algorithms are with people.

Other properties extracted from human segmentations are as follows:
1. Cuts between segments within the same mesh have approximately the same length (i.e length of a cut divided by the average of all cuts is often close to one). This property seems not to be noticed or included in any automatic segmentations.
2. Humans cut meshes into a small number of large (area-wise) sub-meshes and several small ones which corresponds to body and limbs. This suggest that algorithms which aim for equal parts (such as K-mean) may poorly mimic what humans do.

Properties extracted from computer-generated segmentations are as follows:
1. No algorithm outperforms others in all categories. It even seems that algorithms which are designed for a specific type of objects do not necessarily work best in that category.
2. All four evaluation metrics are very consistent with each other and they almost report the same relative performances.
3. The main reason why completely automatic algorithms (with no given parameter) [18, 36] perform worse than the best algorithm is due to the number of segments. If the number of segments is fixed as the number of segments that these algorithms produce, then their performance is similar to the best algorithm. This suggests that it is a good idea for algorithms to include an option for users to manually set the target number of segments.
4. Even when the correct number of segments is given to algorithms, there is a significant difference between the best algorithm and the human generated segmentations.

The relative performances of the mentioned algorithms and their average time consumptions is summarized in Figure 8.

Object Category	Rand Cuts	Shape Diam	Norm Cuts	Core Extra	Rand Walks	Fit Prim	K-Median
Human	1	5	2	7	6	3	4
Cup	1	5	2	3	4	6	7
Glasses	1	4	2	6	7	5	3
Airplane	2	1	4	7	6	3	5
Ant	2	1	3	4	5	6	7
Chair	4	2	1	5	3	6	7
Octopus	4	1	3	2	5	7	6
Table	7	4	1	5	2	3	6
Teddy	1	2	4	3	5	6	7
Hand	1	7	3	4	5	6	2
Plier	2	7	4	1	5	3	6
Fish	3	1	5	2	4	7	6
Bird	1	2	4	3	7	6	5
Armadillo	3	1	5	7	4	2	6
Bust	1	3	6	5	2	4	7
Mech	4	3	1	6	2	5	7
Bearing	2	1	3	7	5	4	6
Vase	1	4	3	2	5	6	7
FourLeg	2	1	6	4	7	3	5
Overall	1	3	2	5	6	4	7

Segmentation Algorithm	Avg Compute Time (s)
Randomized Cuts	83.8
Shape Diameter	8.9
Normalized Cuts	49.4
Core Extraction	19.5
Random Walks	1.4
Fitting Primitives	4.6
K-Means	2.5

Figure 8: Quantitative comparison between seven segmentation algorithms and their average time in seconds [8].

5 Prospective Research Directions

We have surveyed several mesh segmentation algorithms and examined their metrics and selection criteria. In the previous section we point out that currently there are no mesh algorithms that are best in all categories. The search for better and faster algorithms will undoubtedly continue.

Furthermore, research potentials in the field of mesh segmentation can be compared to the field of image segmentation from which various ideas have been borrowed. Looking at the large number of publications and various directions in image segmentation, one can anticipate the same for mesh segmentation.

Some of these possible directions are as follows:

1. Examining the complexity of the problem and algorithms:
As we mentioned in section 3, the mesh segmentation problem can be characterized as an optimization problem. Depending on the desired properties of the resulting segments, different versions of the problem can be defined. The only result achieved for the complexity of the problem seems to be with respect convex partitioning with the minimum number of parts (section 3.1). Even though other versions of the problem appear to be as hard and most studies resort to approximation solutions, an exact complexity result for these problems has not yet been achieved. The main reason for this may lie in the hardness of precisely defining other versions of the problem. Concepts such as meaningful components, natural looking, etc., which are common in the definition of a mesh segmentation problem, are not well-defined. An important research direction is to derive precise definitions for these problems. It is worth noting that almost all segmentation algorithms resort to approximate solutions without proving the hardness of their problem.

The second issue is on the complexity of algorithms themselves. Several of these algorithms use iterations or external procedures in order to carry out their tasks, which makes the exact calculation of their complexity rather difficult. Thus, it is very common for papers to state the time consumption of their algorithm in seconds rather that using complexity notations. A deeper analysis of time and space complexity of existing algorithms may be very useful.

Lastly, even for well-defined versions (such as the minimum convex segmentation problem), approximation algorithms do not guarantee any quality on their results. In contrast to many other problems where approximation algorithms guarantee a solution no worse that a factor of the optimal solution, mesh segmentation algorithms seem to ignore this completely. Finding and proving guaranteed approximation algorithms seems to be a very interesting and yet a neglected direction.

2. Using parallelism in mesh segmentation:
In some applications such as finite element methods, the result of the mesh segmentation is used for parallelism. An intriguing question asks if this process can be done in parallel. This question can be approached in two ways; trying to convert existing algorithms into parallel algorithms (multi region growing algorithms seem like good candidates) or seeking

a new parallel algorithm from scratch. There is little done so far in this field and what has been done is for specific usages [41, 45]. Current extensive usage of computers with multiple processors emphasizes the importance of this direction.

3. Computing the correct number of desired segments:
In section 4, we discuss that the main reason of the relatively lower performance of completely automatic mesh segmentation algorithms lies in their estimation of the number of partitions. Usually the number of segments is not recognized until after the segmentation is performed. Using the benchmark of section 4, one can study human-made segmentations to seek direct relations between the number of segments and characteristics of the model without computing the actual segmentation. A fast algorithm which accurately estimates the number of segments is useful for almost all segmentation algorithms and can further push forward the goal of finding efficient and reliable fully-automatic mesh segmentation algorithms.

References

[1] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation - a comparative study. *Shape Modeling and Applications, International Conference on*, 0:7, 2006.

[2] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22:181–193, 2006.

[3] Richard E. Barlow, Bartholomew D.J, J.M Bremner, and H.D Brunk. *Statistical inference under order restrictions : the theory and application of isotonic regression.* Wiley series in probability and mathematical statistics, no. 8. J. Wiley, London, New York, 1972. Nouveau tirage en 1978.

[4] Halim Benhabiles, Jean Phillipe Vandeborre, Guillaume Lavou, and Mohamed Daoudi. A framework for the objective evaluation of segmentation algorithms using a ground-truth of human segmented 3D-models. In *IEEE International Conference on Shape Modeling and Applications (SMI)*, June 2009.

[5] Matthew Brand and Kun Huang. A unifying theorem for spectral embedding and clustering, 2003.

[6] Scott A. Canann, S. N. Muthukrishnan, and R. K. Phillips. Topological refinement procedures for triangular finite element meshes. *Engineering with Computers*, 12:243–255, 1996.

[7] Bernard Chazelle, David P. Dobkin, Nadia Shouraboura, and Ayellet Tal. Strategies for polyhedral surface decomposition: an experimental study. In *Proceedings of the eleventh annual symposium on Computational geometry*, SCG '95, pages 297–305, New York, NY, USA, 1995. ACM.

[8] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics*, 28(3), 2009.

[9] Siu-Wing Cheng, Tamal K. Dey, and Jonathan Shewchuk. *Delaunay Mesh Generation*. Chapman & Hall/CRC, 1st edition, 2012.

[10] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[11] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, I3D '01, pages 49–58, New York, NY, USA, 2001. ACM.

[12] Natasha Gelfand and Leonidas J. Guibas. Shape segmentation using local slippage analysis, 2004.

[13] Dirk Gerrits, Rene Gabriels, and Kooijmans Peter. A survey of mesh generation techniques. Technical report, Downloaded at dirkgerrits.com/wp-content/uploads/mesh-generation-survey.pdf on August 2013, 2006.

[14] John R. Gilbert, Gary L. Miller, and Shang-Hua Teng. Geometric mesh partitioning: Implementation and experiments. *SIAM Journal on Scientific Computing*, 19(6):2091–2110, 1998.

[15] Aleksey Golovinskiy and Thomas Funkhouser. Randomized cuts for 3D mesh analysis. *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA)*, 27(5), December 2008.

[16] Craig Gotsman. On graph partitioning, spectral analysis, and digital mesh processing. In *In Proc. Intl. Conf. Shape Modeling and Applications (2003*, pages 165–174. IEEE Computer Society, 2003.

[17] Zachi Karni and Craig Gotsman. Spectral compression of mesh geometry. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 279–286, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[18] Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8-10):649–658, 2005.

[19] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics*, 22(3):954–961, 2003.

[20] J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[21] Yu kun Lai, Shi min Hu, Ralph R. Martin, and Paul L. Rosin. Fast mesh segmentation using random walks. In *In ACM Symposium on Solid and Physical Modeling*, 2008.

[22] Guillaume Lavou, Florent Dupont, and Atilla Baskurt. A new CAD mesh segmentation method, based on curvature tensor analysis. *Computer Aided Design*, 37(10):975–987, 2005.

[23] Yunjin Lee, Seungyong Lee, Ariel Shamir, Daniel Cohen-Or, and Hans-Peter Seidel. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des.*, 22(5):444–465, July 2005.

[24] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371, July 2002.

[25] Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang. Decomposing polygon meshes for interactive applications. In *Proceedings of the 2001 Symposium on Interactive 3D graphics*, 2001.

[26] Jyh-Ming Lien, John Keyser, and Nancy M. Amato. Simultaneous shape decomposition and skeletonization. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*, SPM '06, pages 219–228, New York, NY, USA, 2006. ACM.

[27] Rong Liu and Hao Zhang. Segmentation of 3d meshes through spectral clustering. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, PG '04, pages 298–305, Washington, DC, USA, 2004. IEEE Computer Society.

[28] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: Partitioning models into 3D-printable parts. *ACM Transactions on Graphics*, 31(6), 2012.

[29] Alan P. Mangan and Ross T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 1999.

[30] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Automatic mesh partitioning. *Graphs Theory and Sparse Matrix Computation*, 56:57–84, 1993.

[31] P. Moller and P Hansbo. On advancing front mesh generation in three dimensions. *IJNME*, 38:3551–3569, 1995.

[32] M. Mortara, G. Patan E, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Blowing bubbles for the multi-scale analysis and decomposition of triangle-meshes. *Algorithmica*, 38:227–248, 2003.

[33] M. Mortara, G. Patanè, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Plumber: a method for a multi-scale decomposition of 3d shapes into tubular primitives and bodies. In *Proceedings of the ninth ACM symposium on Solid modeling and applications*, SM '04, pages 339–344, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[34] Steven J. Owen. A survey of unstructured mesh generation technology. In *INTERNATIONAL MESHING ROUNDTABLE*, pages 239–267, 1998.

[35] Ariel Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556, 2008.

[36] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Image and Vision Computing*, 24(4):249–259, 2008.

[37] Shymon Shlafman, Ayellet Tal, and Sagi Katz. Metamorphosis of polyhedral surfaces using decomposition. *Comput. Graph. Forum*, 21(3):219–228, 2002.

[38] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 355–362, Washington, DC, USA, 2002. IEEE Computer Society.

[39] Y. Sun, D. L. Page, J. K. Paik, A. Koschan, and M. A. Abidi. Triangle mesh-based edge detection and its application to surface segmentation and adaptive surface smoothing. *IEEE International Conference on Image Processing*, pages 825–828, 2002.

[40] S. Valette, I. Kompatsiaris, and M.G. Strintzis. A polygonal mesh partitioning algorithm based on protrusion conquest for perceptual 3d shape description. *Workshop towards Semantic Virtual Environments*, pages 68–76, 2005.

[41] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. Technical report, Parallel Comput, 1999.

[42] C. Walshaw, M. Cross, and K. McManus. Multiphase mesh partitioning. *Applied Mathematical Modelling*, 25:123–140, 1999.

[43] Kenong Wu and Martin D. Levine. 3d part segmentation using simulated electrical charge distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1223–1235, 1997.

[44] Mark Yerry and Mark Shephard. A modified quadtree approach to finite element mesh generation. *IEEE Computer Graphics and Applications*, 3:39–46, 1983.

[45] Oguz Tosun Ali Haydar Ozer Seren Soner Yusuf Yilmaz, Can Ozturan. Parallel mesh generation, migration and partitioning for the elmer application. Technical report, Partnership for Advanced Computing in Europe, downloaded at http://www.prace-ri.eu/IMG/pdf/ on August 2013.

[46] Hao Zhang and Rong Liu. Mesh segmentation via recursive and visually salient spectral cuts. In *In Proc. of Vision, Modeling, and Visualization*, pages 429–436, 2005.

[47] Yinan Zhou and Zhiyong Huang. Decomposing polygon meshes by means of critical points. In *MMM: Proceedings of the 10th International Multimedia Modelling Conference*, pages 187–195. IEEE Computer Society, 2004.

[48] Emanoil Zuckerberger, Ayellet Tal, and Shymon Shlafman. Polyhedral surface decomposition with applications. *Computers and Graphics*, 26(5):733 – 743, 2002.