Randomized Cuts for 3D Mesh Analysis

Aleksey Golovinskiy and Thomas Funkhouser
Princeton University

Abstract

The goal of this paper is to investigate a new shape analysis method based on randomized cuts of 3D surface meshes. The general strategy is to generate a random set of mesh segmentations and then to measure how often each edge of the mesh lies on a segmentation boundary in the randomized set. The resulting "partition function" defined on edges provides a continuous measure of where natural part boundaries occur in a mesh, and the set of "most consistent cuts" provides a stable list of global shape features. The paper describes methods for generating random distributions of mesh segmentations, studies sensitivity of the resulting partition functions to noise, tessellation, pose, and intra-class shape variations, and investigates applications in mesh visualization, segmentation, deformation, and registration.

Keywords: shape analysis, mesh segmentation

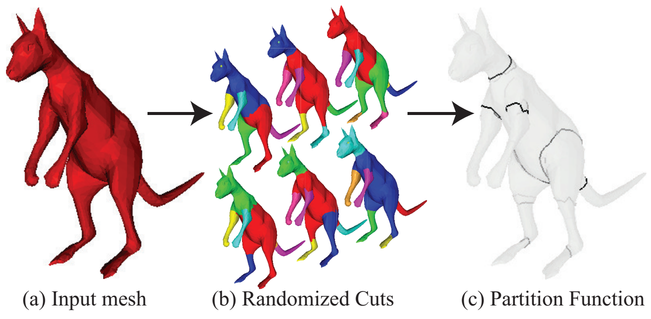


Figure 1: Shape analysis with randomized cuts. Given a 3D mesh as input(a), we sample segmentations from a distribution (b) to generate a *partition function* that estimates the likelihood that an edge lies on a segmentation boundary. This function provides global shape information that is useful for mesh visualization, analysis, and processing applications.

1 Introduction

Shape analysis of 3D surfaces is a classical problem in computer graphics. The main goals are to compute geometric properties of surfaces and to produce new representations from which important features can be inferred.

In this paper, we investigate a new shape analysis method based on *randomized cuts* of 3D surface meshes. The basic idea is to characterize how and where a surface mesh is most likely to be cut by a segmentation into parts.

Our approach is to compute multiple randomized cuts, each of which partitions faces of the mesh into functional parts (Figure 1b). From a set of random cuts, we derive: 1) a *partition function* that indicates how likely each edge is to lie on a random cut (Figure 1c) and 2) a set of the most consistent cuts.

These structures provide global shape information useful for visualization, analysis, and processing of 3D surface meshes. For example, the partition function provides a continuous function for visualization of "chokepoints" (Section 7.1), deformation at joints (Section 7.4), and selection of stable features for surface registration (Section 7.3). The most consistent cuts are useful for finding part boundaries in surface segmentation (Section 7.2).

The paper makes the following research contributions. First, it introduces two new structures for characterization of 3D surface meshes: the *partition function* and the *consistent cuts*. Second, it describes three alternative methods for computing them, based on randomized K-means, hierarchical clustering, and minimum cut algorithms. Third, it provides an empirical analysis of the properties of the partition function, focusing on sensitivity to surface noise, tessellation, pose, and shape variations within an object class. Finally, it demonstrates applications of randomized cuts in mesh visualization, registration, deformation, and segmentation.

2 Related Work

Our work is related to prior work in a diverse set of areas, including shape analysis, mesh segmentation, and randomized algorithms. The following discussion reviews the most relevant previous work.

Shape Analysis: Our goals are motivated by recent work in computer graphics on defining geometric properties for analysis, visualization, and processing of 3D surface meshes. Some classic examples of local shape properties include curvature, slippage [Gelfand and Guibas 2004], accessibility [Miller 1994], saliency [Lee et al. 2005a], multiscale analysis [Mortara et al. 2003], and electrical charge distribution [Wu and Levine 1997]. Recent examples of global shape properties include shape diameter [Shapira et al. 2008], ambient occlusion [Zhukov et al. 1998], spectral analysis [Zhang et al. 2007], and average geodesic distance [Hilaga et al. 2001]. These properties have been used for visualization [Zhukov et al. 1998], segmentation [Shapira et al. 2008], skeletonization [Katz and Tal 2003], matching [Hilaga et al. 2001], and several other applications. Our partition function adds to this list: it is a global shape property that measures the probability that a surface point lies on a segmentation boundary.

Mesh Segmentation: The methods in this paper leverage recent work on segmentation of 3D meshes into parts. A wide variety of algorithms have been proposed for this problem, based on convex decomposition [Chazelle et al. 1997], watershed analysis [Mangan and Whitaker 1999], K-means [Shlafman et al. 2002], hierarchical clustering [Garland et al. 2001; Gelfand and Guibas 2004; Inoue et al. 2001], region growing [Zuckerberger et al. 2002], mesh simplification [Li et al. 2001], spectral clustering [Liu and Zhang 2004], fuzzy clustering and minimum cuts [Katz and Tal 2003], core extraction [Katz et al. 2005], critical point analysis [Lin et al. 2004], tubular primitive extraction [Mortara et al. 2004], primitive fitting [Attene et al. 2006b], random walks [Lai et al. 2008], Reeb graphs [Antini et al. 2005], snakes [Lee et al. 2005b], and other methods (see [Agathos et al. 2007] or [Shamir 2006] for a recent survey). Our randomized cuts approach is synergistic with these methods in two ways. First, it uses (randomized versions of) existing mesh segmentation algorithms to make cuts when computing the partition function. Second, it utilizes the partition function to score candidate cuts in a new hierarchical segmentation algorithm. As such, our work benefits from improvements to mesh segmentation algorithms and vice-versa.

Random Cuts: Our approach follows a vast amount of prior work on randomized algorithms for graph partitioning in theory and other subfields of computer science. Perhaps the most relevant of this work is the algorithm by Karger and Stein [Karger and Stein 1996], which utilizes a randomized strategy to find the minimum cut of a graph. The main idea is to generate a large set of randomized cuts using iterative edge contractions, and then to find the minimum cost cut amongst the generated set. Our paper differs in that we use randomized cuts to produce a partition function and to generate a set of scored cuts (rather than finding a single minimum cut).

Typical Cuts: Other authors have used randomized algorithms for graph analysis. For instance, [Gdalyahu et al. 2001] generated multiple cuts in a graph representing an image with randomized iterations of hierarchical clustering to produce a similarity function indicating the probability that two nodes reside in the same cluster. The probabilities were used to segment the image with "typical" cuts, by constructing connected components of the graph formed by leaving edges of probability greater than 0.5. The result is both a pairwise similarity function for pixels and an image segmentation. Our work takes a similar approach in that we use randomized clustering to explore the likelihood that nodes lie in the same cluster (more precisely, that edges lie on the boundaries between clusters). However, we investigate a several randomized clustering strategies (finding that the hierarchical algorithm of [Gdalyahu et al. 2001] does not perform best in our domain), produce a partition function and a set of consistent cuts for 3D meshes, and investigate applications for 3D mesh visualization, analysis, and processing.

3 Overview of Approach

In this paper, we investigate the use of random cuts for shape analysis of 3D meshes. The general strategy is to randomize mesh segmentation algorithms to produce a function that captures the probability that an edge lies on a segmentation boundary (a cut) and to produce a ranked set of the most consistent cuts based on how much cuts overlap with others in a randomized set.

This strategy is motivated by three factors. First, no single algorithm is able to partition every mesh into meaningful parts every time [Attene et al. 2006a]. Second, existing segmentation algorithms often have parameters, and different settings for those parameters can significantly alter the set of cuts produced. Third, segmentations produced by different algorithms and parameter settings often cut many of the same edges [Attene et al. 2007]. Thus, it could be useful to combine the information provided by multiple discrete segmentations in a probabilistic framework, where the output is not a single discrete segmentation, but rather a continuous function that captures the likelihood characteristics of many possible segmentations.

For instance, consider the example shown in Figure 1. The images in Figure 1b show discrete segmentations of a kangaroo produced by a randomized hierarchical clustering algorithm (as described in Section 5.2). Although every segmentation is different, some boundaries are found consistently (e.g., along the neck, cutting off the legs, arms, tail, etc.). These consistencies are revealed in the partition function in Figure 1c, which shows for each edge the probability that it lies on a segment boundary. We believe that this continuous function reveals more information about the part structure of the kangaroo than does any one of the discrete segmentations.

More formally, we define the *partition function* $P(e,S)$ for edge e of a 3D mesh with respect to a distribution of segmentations S to be the probability that e lies on the boundary of a random segmentation drawn from S. From this partition function, we define the *consistency*, $P(S_i)$, of a cut, S_i, within a distribution S as the length-weighted average of the partition function values of its edges, and we define the *most consistent cuts* as the set of cuts with highest consistency.

This formulation is similar to the one used to find typical cuts for image segmentation [Gdalyahu et al. 2001], but generalizes it in five main ways. First, the construction combines multiple segmentation algorithms (not just hierarchical clustering) and it allows randomization of any variable guiding those algorithms (not just which edge to contract). Second, it produces a partition function on edges (rather than a similarity function on pairs of nodes). Third, it produces a continuous score for every random cut (not just a discrete set of cuts whose edges all have probability greater than 0.5). Fourth, it provides a natural way to measure the consistency of cuts. Finally, it suggests a number of applications that utilize continuous functions on meshes (not just segmentation). Our main contributions are investigating the space of randomization strategies and mesh processing applications within this general formulation.

4 Process

The input to our processing pipeline is a 3D surface mesh, M (Figure 1a), and the outputs are: 1) a randomized sampling of segmentations, $\hat{S}$ (Figure 1b), 2) a partition function estimate, $P(e,\hat{S})$, for every edge e of the mesh with respect to $\hat{S}$ (Figure 1c), 3) a consistency score for every segmentation indicating how consistent its cuts are with others in $\hat{S}$, and 4) a ranked set of most consistent cuts. The processing pipeline proceeds in four main steps:

1. Graph construction: The first step produces an edge-weighted graph G representing the input mesh. We follow a standard approach to this problem, building the dual graph of the input mesh, where nodes of the graph correspond to faces of the mesh, and arcs of the graph correspond to edges between adjacent faces in the mesh. Because some algorithms seek to minimize traversal distances across a graph, while others minimize graph cuts, we associate two weights to each graph arc: a traversal cost, and a cut cost. The weights are created such that low-cost cuts in the graph correspond to favorable segmentation boundaries in the mesh, and low-cost traversal paths in the graph occur between points on the mesh likely to be in the same functional parts. Similarly to [Funkhouser et al. 2004], we assign a concavity weight to each mesh edge as follows: if θ is the exterior dihedral angle across an edge, we define a concave weight $w(\theta) = \min((\theta/\pi)^{\alpha}, 1)$, which is low for concave edges, and 1 for convex ones (we use $\alpha = 10$ in all experiments and examples in this paper). We form cut costs by multiplying the mesh edge length by $w(\theta)$, and traversal costs by dividing the distance between face centroids by $w(\theta)$. The cut cost of a mesh segment then is its perimeter, weighted to encourage concave boundaries, and the traversal costs represent geodesic distances on a mesh, weighted to make travel through concave edges longer.

2. Random cuts: The second step generates a randomized set of K-way segmentations, where each segmentation partitions faces of the mesh into disjoint parts. For this step, we investigated several algorithms (including K-means, Hierarchical clustering, and Mincuts), and we randomized them with several different strategies. This step is the main focus of our study, and thus we defer details to Section 5. For the purpose of this section, it suffices to say that the output of this stage is a scored set of segmentations, where each segmentation S_i provides a set of edges C that are "cut" by the segmentation (lie on the boundary between segments), and the score indicates the "quality" of the segmentation (e.g., the normalized cut value).

3. Partition function: The third step estimates the partition function for each edge. Given a set of K-way segmentations, $\hat{S}$, randomly sampled from S, this can be done trivially for each edge e by summing the weights of all segmentations in $\hat{S}$ that contain a boundary on e and dividing the sum by the total weight of all segmentations in $\hat{S}$. If all segmentations have equal weight, then the result is an estimate of the probability that an edge is cut by a segmentation in S.

4. Cut consistency: The final step provides a consistency score for every segmentation S_i in $\hat{S}$. It is computed by taking a weighted average of the partition function values along edges on the boundaries of segments in S_i (where averaging weights are proportional to edge lengths). The M segmentations with highest scores (possibly all of them) form a set that we call the *most consistent cuts*.

The main focus of our project is on methods for generating random segmentations, analyzing properties of the resulting partition functions, and demonstrating applications in computer graphics. The following sections address these three issues, respectively.

5 Methods

The key step of our process is to define a randomized process that creates a distribution of mesh segmentations and to sample that process to compute properties of the distribution (the partition function). Several strategies are possible. First, random variables could determine which algorithmic strategy should be used to generate segmentations (e.g., K-means vs. hierarchical clustering). Second, they could determine parameters for specific algorithms (e.g., the number of segments in K-means). Third, they could guide choices made within a segmentation algorithm (e.g., the order of hierarchical edge contractions [Karger and Stein 1996]). Finally, they could provide randomized variation to the input graph (e.g., randomized edge weights). Of course, combinations are possible as well. Each combination of random variables produces a distribution of cuts, and therefore a different partition function and set of most consistent cuts. In this section, we investigate a range of strategies for randomizing cuts and describe details of our implementations.

5.1 K-Means

K-means is a popular algorithm for clustering, and its variants have been used for graph partitioning and mesh segmentation. For example, [Shlafman et al. 2002] used K-means to decompose faces of a mesh into parts as follows. First, K seed faces were chosen to represent segment centers with a deterministic greedy process that maximizes their pairwise distances. Then, the algorithm alternated between assigning faces of the mesh to the segment represented by the closest seed face (according to traversal costs) and recomputing the seed faces to minimize the sum of distances to faces in their segments.[1] This process was iterated until convergence.

[1]In our implementation, seed faces are updated to be the ones furthest from their segment boundaries, a slight change from [Shlafman et al. 2002] that we found gave better results for our graphs.

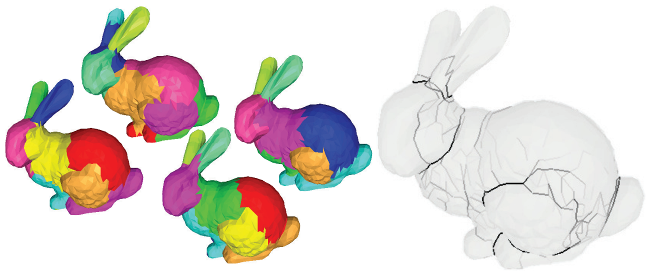

(a) Random Segmentations (b) Partition Function

Figure 2: Sample randomized segmentations produced with K-means (a) and the resulting partition function (b).

We have experimented with this approach within our framework by randomizing the set of seed faces chosen during the first step of the algorithm. That is, for each new segmentation sampled from the random distribution, we simply select K faces randomly from the mesh and then iterate K-means to convergence. After many randomized segmentations, we generate the partition function value for each edge by computing the fraction of segmentations for which that edge lies on a segment boundary.

Figure 2 shows an example partition function produced for the Stanford Bunny with this randomized K-means algorithm. The left image shows 4 of 1200 segmentations generated with K-means initialized with random sets of 10 seeds. The right image shows the resulting partition function. Note the dark lines indicating strong consistency in the cuts along the neck and front of the legs, while the cuts at the base of the ears and back of the legs do not always agree, and therefore appear lighter.

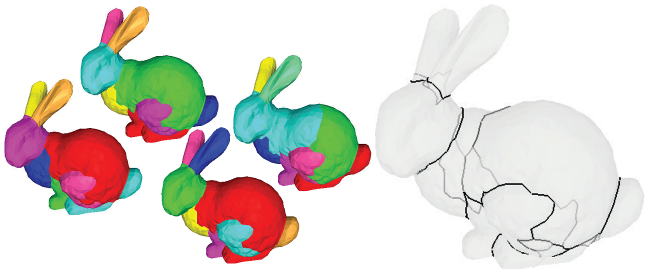
Since the value of K given as input to the K-means algorithm dictates the number (and therefore size) of parts generated in each segmentation, there is a family of possible partition functions that could be produced by this algorithm. For simplicity, we provide results for only one setting, $K = 10$, as it provides a good trade-off between parts that are too big and ones that are too small. Alternatively, it would be possible to randomize K for different segmentations. However, we expect that approach would provide worse results (it would add poor segmentations to the distribution).

5.2 Hierarchical Clustering

Hierarchical clustering is another popular segmentation algorithm. In the context of surface meshes, the process starts with every face in a separate segment, and then merges segments iteratively in order of a cost function until a target number of segments (K) has been reached or some other termination criterion is met. Typical cost functions include the minimum cut cost of any edge between two segments (single-link clustering), the total cost of the cut between two segments [Gdalyahu et al. 2001; Karger and Stein 1996], and the Normalized Cut cost [Shi and Malik 2000].

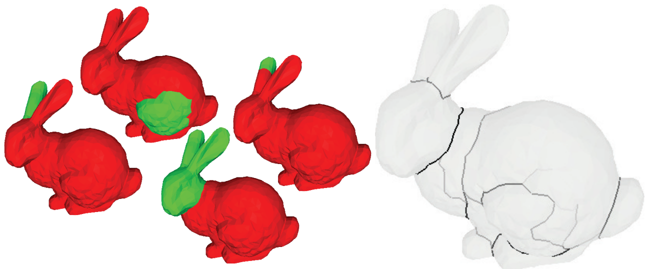
In our work, we use a slightly modified Normalized Cut cost to guide the random order in which segments are merged. Specifically, for every pair of adjacent segments, we compute an area-weighted version of the Normalized Cut cost[2] for the segmentation that would result if the segments were merged. Then, we set

[2]The area-weighted cost function sums for all segments the ratio of the segment's cut cost to its area. We chose this cost function over the traditional Normalized Cuts (which divides by the cost of all edges in a segment) to avoid dependence on mesh tessellation.



(a) Random Segmentations (b) Partition Function

Figure 3: Sample randomized segmentations produced with Hierarchical Clustering (a) and the resulting partition function (b).



(a) Random Segmentations (b) Partition Function

Figure 4: Sample randomized segmentations produced with the MinCut algorithm (a) and the resulting partition function (b).

the probability of selecting every pair to be a function of that area-weighted Normalized Cut cost. Specifically, to select a pair of segments to merge at each iteration, we map the differences in area-normalized cut associated with contracting each graph arc to [0, 1], raise each value to the power of $(1/r)$ where r is a randomization parameter, and choose an arc to contract with that probability. This way, as r approaches 0, this algorithm approaches its deterministic version. We found that values on the order of $r = .02$ produce a reasonable tradeoff between quality and sufficient randomness in the resulting segmentations. As in K-means, the desired number of segments is an input and controls the size and scale of the parts.

Figure 3 shows the partition function produced by running the randomized hierarchical clustering algorithm while greedily minimizing the area-weighted Normalized Cut cost to produce $K = 10$ segments. As in Figure 2, the left image shows four example segmentations, while the right image shows the resulting partition function. Note that boundaries at the top of the legs and front of the tail are sharper than they are for K-means (in Figure 2). Since the hierarchical algorithm based on area-weighted normalized cuts explicitly favors segmentations with boundaries on concave edges, it is more likely to produce consistent cuts in those areas. Note also that that K does not have to match the "natural" number of segments to produce a quality partition function: the randomized nature of the algorithm alternates between cuts with similar costs, such as cutting off one leg versus another, and therefore both are reflected in the partition function.

5.3 Min Cuts

Finding minimum cuts is a third possible segmentation strategy. The general approach is to select a set of K seed nodes and then to find the minimum cost cut (mincut) that partitions the seeds. This is a classical network flow problem with well-known polynomial time solutions for $K = 2$, and approximation algorithms for $K > 2$. However, in order to avoid trivial cuts that partition one face from the rest of the mesh and similar small-area segments, the mincut problem must be modified with constraints and/or penalties to favor nearly equal-area segments. Towards this end, [Katz and Tal 2003] constrained cuts to lie within a "fuzzy" area, where the relative geodesic distance, $d = min(dist(e,A), dist(e,B))/(dist(e,A) + (dist(e,B))$ between an edge e and seeds, A and B, was greater than $0.5 - \varepsilon$. While this approach avoids the main problem of making trivially small cuts, it can produce cuts that lie on the boundaries of the fuzzy area (e.g., both examples in Figure 3 of [Katz and Tal 2003]), which may do not lie on natural seams of the mesh.

To address this problem, we make a small modification to the minimum cut algorithm of [Katz and Tal 2003]. For every set of seeds, X, we adjust the weights on edges of the graph with a penalty function, $W(e,X)$, that gradually falls off with distance from the closest seed. In our current implementation, $W(e,X)$ is a piecewise linear function that has a constant high value of W_{high} for $d <= x_1$, decreases linearly to 1 at $d = x_2$, and then remains 1 for $d >= x_1$. This approach encourages cuts at larger distances from seeds, but does not enforce hard boundaries on the fuzzy area (within our framework, the method of [Katz and Tal 2003] is equivalent to setting $x_1 = x_2 = .5 - \varepsilon$). For simplicity sake, we control the coefficients and cutoffs for the distance penalty function with a single parameter, s ($W_{high} = 400s + 1$, $x_1 = s$, and $x_2 = 3s$), which roughly controls the "size" of parts to be generated by the segmentation.

We use this MinCut algorithm to generate a set of cuts by randomizing the selection of seed faces. In our current implementation, we consider only cuts between two seeds at a time in order to leverage fast polynomial algorithms for finding $s - t$ mincuts. In each randomized iteration, our algorithm proceeds by randomly selecting two faces, s and t, modulating the original weights on edges of the graph by $W(e,s,t)$, and then finding the minimum cut separating s and t in the modified graph using the Edmonds-Karp algorithm for finding maxflows [Edmonds and Karp 1972]. This process is iterated to generate a sampled set of cuts from which the partition function and most consistent cuts structures are derived.

Figure 4 shows the partition function produced by this process for 400 iterations of the MinCut algorithm with $s = 0.05$. As in Figures 2 and 3, the left image shows four random segmentations produced by the MinCut algorithm when randomly seeded with different sink and seed faces (for example, the bottom-right segmentation came from a source on the face and a sink on the body, while the bottom-left came from a source on the ear and a sink on the face). Even though the randomized segmentations produced only two segments each, and not all of the 2-way segmentations find meaningful parts (e.g., the top-right segmentation cuts off only part of the ear), they cumulatively reveal the decomposition of the bunny into its natural set of parts, as shown in the partition function shown on the right. As you can see, the cuts produced with the MinCut algorithm are even more consistent (darker and sharper lines in the partition function) than the ones generated with Normalized Cuts.

Figure 5 shows how the partition function reveals features of different size as s is varied: the ears of the dog and mouth of the horse are revealed for $s = 0.01$, and large-scale features such as the cuts through the bodies are revealed at $s = 0.05$. For some applications, it may be useful to created a collection of partition functions for a range of values of s, and treat them as a feature vector for each edge. For others, it may make sense to randomize s within a range. For the experiments in this paper, however, we simply use $s = 0.05$.

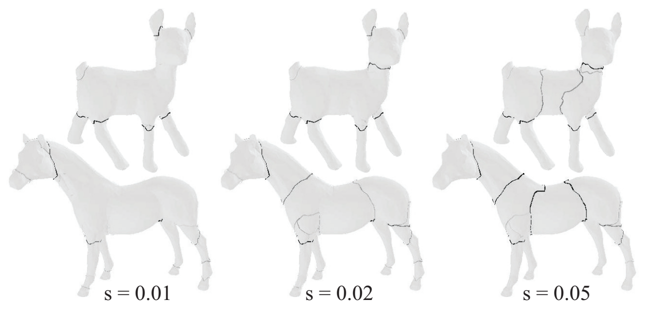


Figure 5: Comparison of partition functions generated with the MinCut algorithm using different settings of the *s* parameter, which loosely controls the "sizes" of parts cut off in random segmentations.

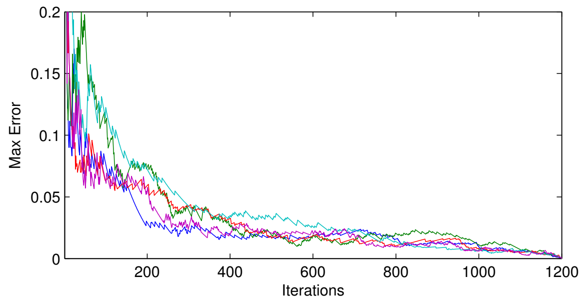


Figure 6: Maximum error in partition function versus iterations for five example models.

6 Experimental Results

We have implemented the algorithms of the previous section and incorporated them into the pipeline for computing the partition function and consistent cuts described in Section 4. In this section, we show examples for many types of meshes, analyze compute times, and investigate sensitivities to noise, tessellation, articulated pose, and intra-class variation. Our goal is to characterize the speed, stability, and practicality of the partition function as a shape analysis tool.

Due to space limitations, it is not possible to do a detailed analysis for all the possible randomized cut strategies described in the previous section. So, in this section, and for the remainder of the paper, we will only discuss results generated with the MinCut algorithm using $s = 0.05$. These results are representative of what is possible with randomized cuts, but possibly not as good as what could be achieved by tuning the algorithm for specific applications.

6.1 Compute Time

Our first experiment investigates the compute time and convergence rates of our randomized process for computing partition functions. We performed a timing test on a Macintosh with a 2.2GHz CPU and 2GB of memory using 5 meshes from Figure 7 (screwdriver, mask, human, pronghorn, and bull). For each mesh, we first decimated it to 4K triangles with QSlim [Garland and Heckbert 1997], and then generated random segmentations using the MinCut algorithm until the partition function stabilized. We measure convergence rates by reporting the *maximum* error for each iteration with respect to the final partition function value (i.e., the maximum error for any edge – RMSD values would be orders of magnitude smaller).

A plot of error versus iteration appears in Figure 6. We can see that after about 400 iterations, the maximum error for any edge in any of the 5 models is below 5% (this is the number of iterations used to make most images in this paper). The algorithm took .6 seconds to run per iteration, and thus an average of 4 minutes were taken to create a stable partition function for each of these models. Both the timings and convergence rates do not vary much between models of similar triangle counts, but they do increase with $O(VE^2)$ for models with N faces and E edges, as we execute Edmonds-Karp to find a mincut for every iteration.

While the computational complexity of the MinCut algorithm grows superlinearly with triangle count, this is not a practical problem for our system, since we aim to reveal large-scale part structures of a mesh, which are apparent at low triangle counts. We strike a conservative balance between mesh resolution and compute time by decimating all meshes to at most 4K triangles, which contain more than enough resolution to describe parts, but allow reasonable compute times.

6.2 Examples

Our second experiment investigates the qualitative properties of the randomized cuts generated with our methods. We do this by computing and visualizing the partition function for a wide variety of examples.

Figure 7 shows a visualization of the partition function computed for meshes representing animals, humans, faces, organs, and tools. From these examples, we see that the partition function generally favors edges that: 1) lie on cuts that partition the mesh with few/short/concave edges, 2) lie on cuts that partition the mesh into nearly equal size parts, and 3) do not lie nearby other more favorable cuts. The first property is a direct result of using the MinCut algorithm to produce segmentations (the cut cost favors few/short/concave edges). The second comes from both penalizing edges close to the source and sink with $W(e,X)$ and random sampling of mincut sinks and sources (edges near the center of the mesh are more likely to lie "between" randomly chosen sources and sinks). The third is a combined result of mincuts and random sampling – an edge will only get votes if it has the least cost among cuts that separate the source and sink. So, a very favorable cut will "dominate" nearby cuts whose cost is not as low.

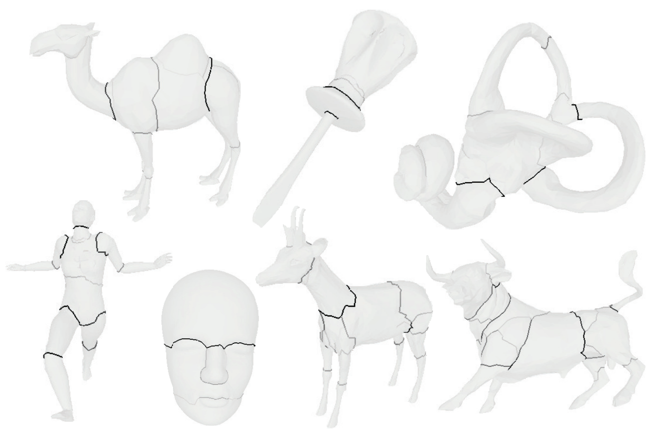


Figure 7: Partition function examples.

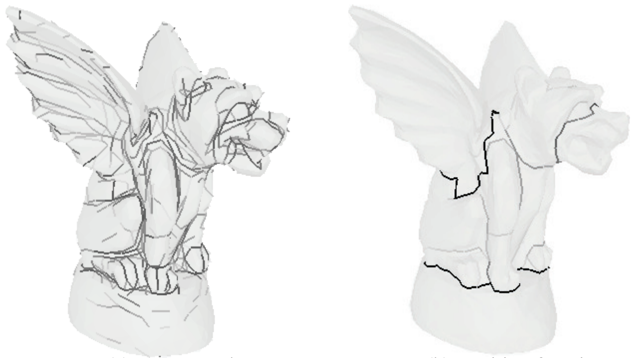

(a) Input Graph (b) Partition function

Figure 8: Comparison of graph edge concavity weights (a) vs. partition function (b). Note that the edge weights (left) capture only local concavity features, while the partition function captures global shape properties.

Figure 8 provides a comparison of the partition function (right) with the original edge concavity weights computed from dihedral angles (left) for a Gargoyle. The main difference is that the partition function captures *global* properties of the mesh. As a result, not all concave edges have large partition function values, and vice-versa. For example, for the Gargoyle, the partition function value is highest on edges that lie along the junctions of the wings-to-body, head-to-body, and feet-to-stand. In some cases, convex edges have high partition function values because they lie on a favorable seam determined by other concave edges (e.g., on the top of the shoulder), while some concave edges have low partition function values because they do not lie on a global seam (e.g., folds of feathers in the wings), or because they do not cut off a large part (junctions of ear-to-head), or because they lie near other global seams (e.g., top of the foot). The global nature of the partition function makes it more useful for applications that require stable, large-scale shape features.

6.3 Sensitivity to Pose and Intra-Class Variation

Our next experiments investigate the sensitivities of the partition function to articulated pose and variations of shapes within the same class of objects. Since edge weights on the dual graph used to generate random segmentations are determined directly from dihedral angles (which vary with changes to pose and between instances of the same class), one might wonder how robust our methods are to variations of this type.

To test sensitivity to pose, we computed the partition function and consistent cuts for seven poses of a horse provided by [Shapira et al. 2008] and twenty poses of the Armadillo provided by [Giorgi et al. 2007]. As can be seen in Figure 9, the gross characteristics of the partition functions are fairly stable across different poses – the strongest cuts appear consistently along the neck, tail, face of the horse and at the key joints of the Armadillo (knees, thighs, shoulders, etc.). Even though the dihedral angles of individual edges vary with pose, the global properties of the most consistent cuts generally remain stable.

However, there are exceptions. Figure 9 clearly shows variations in the cuts across the torso of the Armadillo. In this case, the torso is nearly symmetric, and diagonal cuts across it have nearly equal costs. Thus, small perturbations to the edge weights can cause a different global cut to be chosen consistently. We believe that this effect can be ameliorated somewhat by randomization of graph edge weights, but experimentation with this strategy is future work.

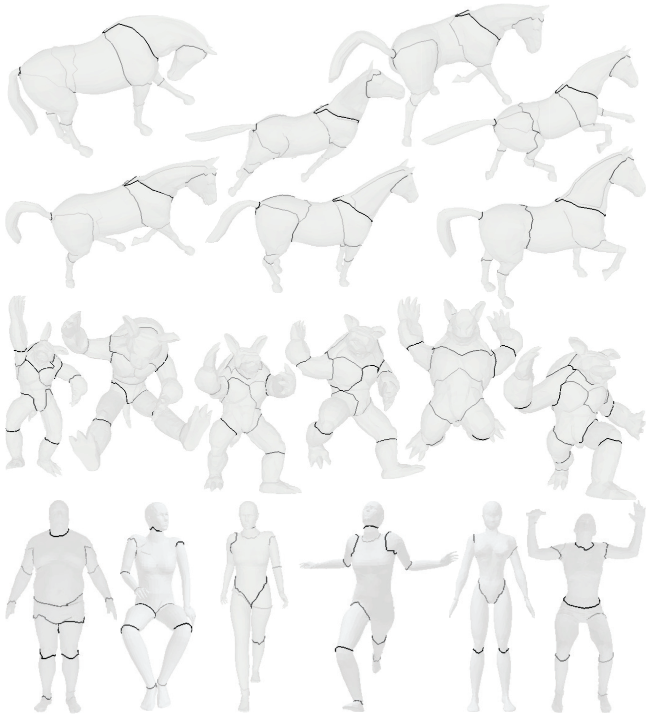


Figure 9: Partition function for different poses (horse and armadillo) and instances within the same class (humans). Note that many of the prominent cuts remain consistent despite considerable variability in body shape.

To investigate variation across objects within the same class, we computed and compared the partition functions for all 400 meshes in all 20 classes of the Watertight Data Set of the SHREC benchmark [Giorgi et al. 2007]. We find that examples within most classes in this data set have similar partition functions. In particular, the most consistent cuts amongst instances of the airplane, ant, armadillo, bird, chair, cups, glasses, hand, octopus, plier, and teddy classes are very similar, while those of the bust, human, mech, spring, and table have greater variation. Representative results are shown for the human class in bottom row of Figure 9. While many of the prominent cuts remain consistent despite considerable variability in body shape and pose (e.g., neck, shoulders, thighs, knees, etc.), there are certainly cases where a large concave seam appears in some objects, but not others (e.g., there is a strong concave waist in three out of the six examples shown). These results suggest that many, but not all, consistent cuts are stable across instances within the same class.

6.4 Sensitivity to Noise and Tessellation

Our fourth experiment studies whether the partition function is sensitive to noise and tessellation of the mesh.

To study sensitivity to noise, we calculated the partition function for a series of increasingly noisy meshes (vertices were moved in a random direction by a Gaussian with the standard deviation of σ times the average edge length). Three meshes with increasing noise (red) and the resulting partition functions (gray) are shown from left-to-right for two objects in Figure 10. These images show that the strong consistent cuts are largely insensitive to noise (e.g., neck), but that weaker cuts at the extremities are diminished (e.g., legs). The reason is as follows. Strong consistent cuts mainly appear at the boundaries between large, nearly-convex parts. Since they are

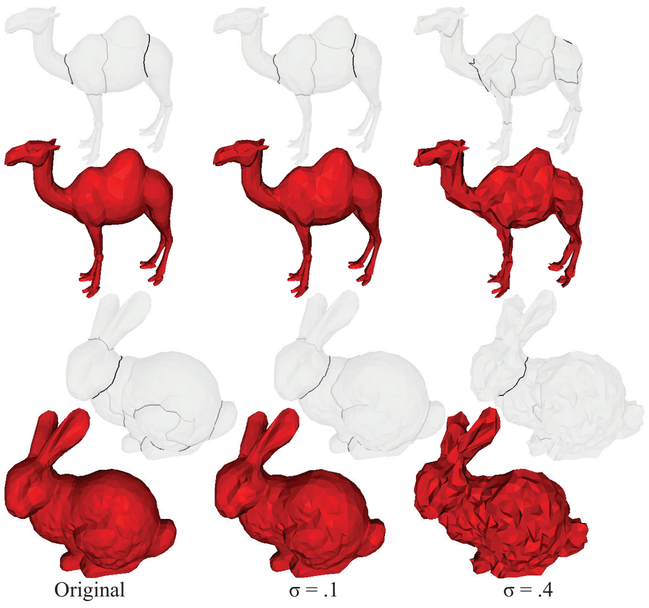


Figure 10: Partition function on noisy meshes: random vertex displacement of standard deviation σ times the average edge length has been applied. Note that the partition function degrades only with high noise, and even then the main cuts remain.

the results of votes by many source-sink pairs selected from opposite sides of a part boundaries, they remain stable as noise is added to the mesh – i.e., there is still a lowest-cost cut that separates the large parts, and that cut is found consistently even in the presence of noise. On the other hand, for boundaries between smaller parts near the extremities, where the partition function is weaker, fewer votes are cast for the cut, and thus new low-cost cuts along concave seams created by gross vertex displacements receive as many votes as they do, and their relative strength is diminished. Overall, we find that the gross structure of partition functions is stable within a moderate range of noise (e.g., less than 10% of average edge length).

To study sensitivity to tessellation, we computed partition functions for a camel model decimated to several resolutions with QSlim [Garland and Heckbert 1997]. Figure 11 shows the results. Although decimation changes the graph representation of the surface considerably (increasing dihedral angles), the most consistent cuts remain largely unchanged (they can even be found after decimation to 250 faces). The exact placement of consistent cuts can vary slightly (e.g., on the neck), and extra cuts sometimes appear as the mesh is decimated and concavities become more exaggerated (e.g., around the hump). However, the original structure of cuts persists for moderate decimations. One concern may be that sharp features are less prominent at higher resolutions. As the zoomed images in Figure 11 show, the cuts are still found consistently as model resolution is increased. Of course, our algorithm assumes that segment boundaries are located along mesh edges, so an adversarial triangulation (such as one where edges cut across a natural boundary) or simplification so extreme that segments are broken (such as in the last image of Figure 11) would pose challenges to our method (and to other graph-based mesh analysis methods).

6.5 Comparisons to Alternative Methods

Our final analysis compares the proposed partition function to other functions previously used for revealing natural decompositions of graphs. The most obvious alternative is the work of [Gdalyahu et al. 2001], which computes Typical Cuts. Their method is a randomized hierarchical clustering, where the probability of merging two segments is proportional to cost of the cut between them. We have implemented this approach and find that it tends to create only very large and very small segments for our graphs (single faces whose cut costs are low get isolated). So, in our comparison, we stop contracting segments when 500 are still remaining in each randomized iteration (which we find gives the best results). A partition function calculated this way is shown in Figure 12b.

Another alternative is spectral embedding. Spectral segmentation methods embed the input graph in a lower-dimensional subspace, and segment in that subspace. Distances in these spectral embeddings can be used for a partition function. For example, the method of [Yu and Shi 2003] creates a segmentation into n segments by first embedding the input graph into an n dimensional space, and then projecting it onto a unit sphere. We can use distances between graph nodes on that sphere to create a function similar in concept to a partition function (shown in Figure 12c).

Comparing results, the partition function more clearly highlights the cuts between natural parts (Figure 12a). Although the level of darkness is different in the three images, the second two have been tuned to reveal as much of the part structure as possible, and adjusting visualization parameters does not reveal the part structure as well as the partition function does.

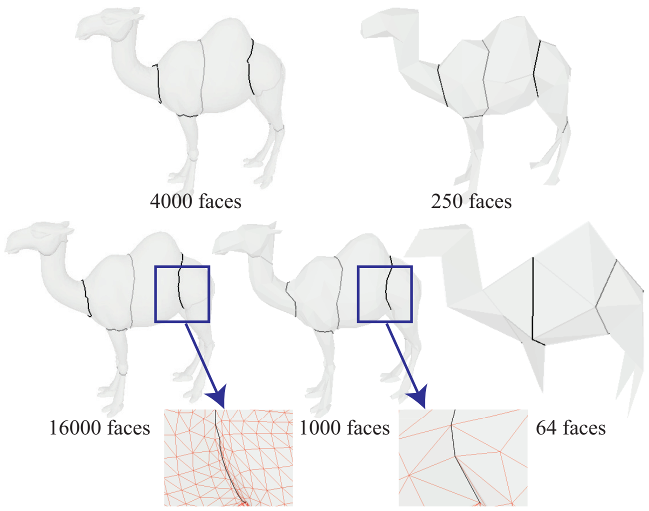


Figure 11: Partition function on a mesh decimated to varying resolutions. Although decimation alters the meshing considerably, the gross structure of the partition function remains largely unchanged until decimation becomes extreme. The two zoomed views show that the partition function remains consistent at the smoother segment boundaries of the 16K mesh (compared to the 1K mesh).

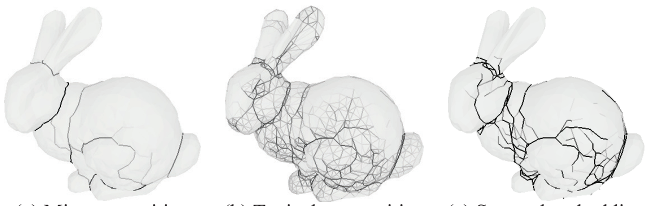


Figure 12: The partition function created by our method (a), compared with Typical Cuts (b), and spectral embedding (c).

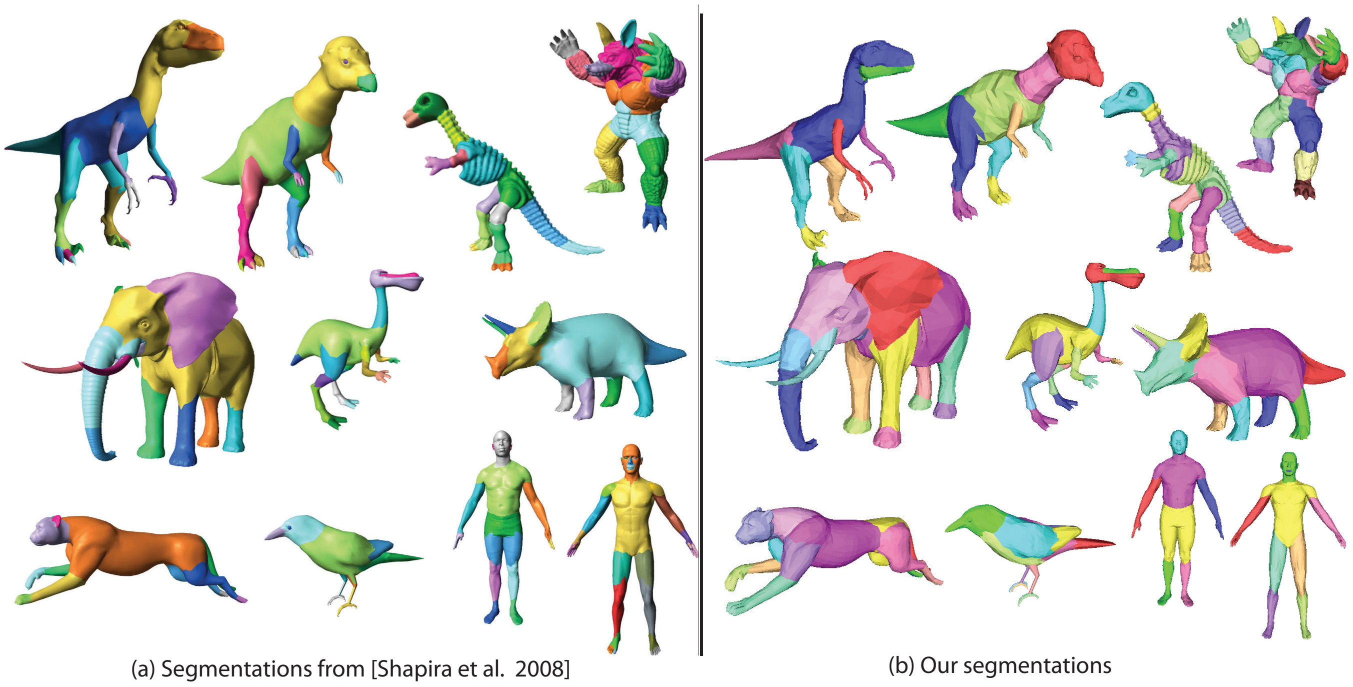


Figure 13: Comparison with [Shapira et al. 2008]. (a) Figure 11 from [Shapira et al. 2008] (reproduced with permission). (b) Our results. None of the segmentations are perfect. Some of their segmentations are better, and some of ours are better; overall, the quality is similar between the two methods.

7 Applications

Randomized cuts are a basic shape analysis tool that may be useful in a wide range of application domains. In this section, we investigate four applications in computer graphics.

7.1 Visualization

An obvious application of the partition function is visualization. In the figures shown in previous sections of this paper, meshes are drawn with diffuse shading in light gray, with lines superimposed over edges based on the partition function (darker lines represent higher partition function values). In most cases, this visualization provides a sense for the whole shape (due to the light gray shading), while highlighting the seams between large parts (due to the dark lines), and thus perhaps it conveys the large-scale parts structure of the surface more clearly than diffuse shading alone. Of course, it can also be useful to display a sequence of the most consistent cuts, and/or provide the user with an interactive tool to explore a ranked set of consistent cuts to better understand the structure of a surface. While we have experimented with these methods for our own understanding of partition functions, we have not investigated the full potential of visualization of surfaces based on partition functions and consistent cuts, which is a topic for future work.

7.2 Segmentation

Segmentation is an important pre-processing step to many algorithms. Many methods exist to segment models; some of these are described in Section 2, and some of their randomized variations are described in Section 5. The partition function and ranked cuts represent a consensus of which edges and cuts are more likely to participate in a segmentation, and are therefore useful to segment a model.

There are several ways the partition function can be used to segment models. The idea suggested in Typical Cuts [Gdalyahu et al. 2001] is to define as segments those parts of the mesh that are connected with partition function edges less than some threshold (they use .5). While this has the benefit of simplicity and a natural stopping criterion, it has two major drawbacks when applied to mesh segmentation. First, the desirable cut to make within a segment should depend on the surrounding segments. For example, there are frequently cuts of similar probabilities that are close to one another, often overlapping, and rather than segment the narrow slivers between these close cuts, it would be better to recalibrate the partition function within the chosen segment. Second, it is impractical to compute the partition function globally for all levels of detail, since most of the computation of calculating a partition function on the entire mesh is spent on finding large, global cuts.

This suggests a hierarchical algorithm: compute the partition function on the current segment (starting with the entire mesh), split the segment into child segments with the most consistent cut, and recurse. Re-computing the partition function in each segment discourages cuts adjacent to existing segment boundaries, and focuses computation on cuts within the segment. However, the resulting probabilities of cuts within child segments are not normalized with respect to one another, and so we use the area-normalized cut cost (see Section 5.2) to choose the order of the recursion. The full algorithm is: cut the current segment with the most consistent cut, compute the partition function on the child segments and propose splits of each child segment, and put these proposed splits into a global priority queue ordered by the area-normalized cut of the resulting segmentation. The algorithm takes as input the desired number of segments, and uses that as its termination criterion.

Figure 13 has our segmentation results for a variety of meshes (b), and compares them to the results of [Shapira et al. 2008] (a). The partition function finds segment boundaries on these meshes through a combination of their being partially concave, and/or be-

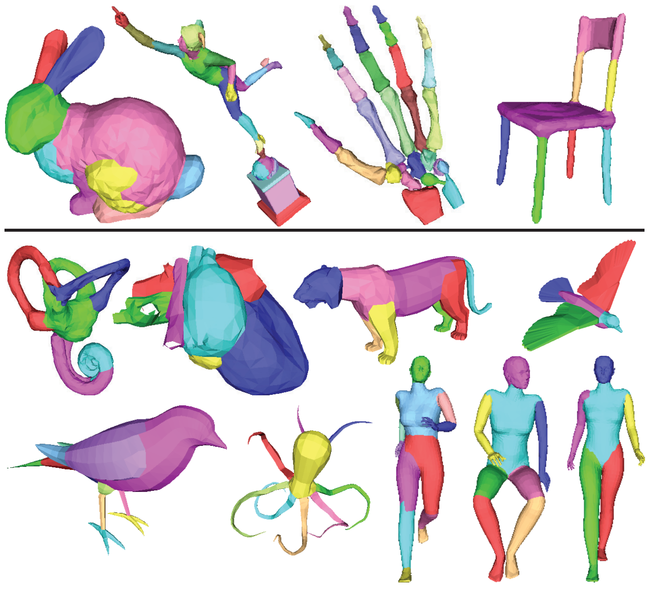


Figure 14: More segmentation examples. The models in the bottom two rows appeared in the segmentation comparison study of [Attene et al. 2006a] (in their Figures 7, 8, 10, and 11). The quality of our segmentations is similar to that of the best methods in their survey.

ing closer to the "middle" of previous segments. Some of the segmentations of [Shapira et al. 2008] are better, and some of ours are better; overall, the quality is similar between the two methods.

Figure 14 shows segmentations of more shapes. The models in the bottom two rows were chosen to provide a direct comparison to the segmentation survey of [Attene et al. 2006a] (to save space, we do not reprint their results, but direct the reader to their Figures 7, 8, 10, and 11). Again, we find that the quality of our segmentations appears similar to the best of prior work. However, to be fair, we have not included an automatic criterion for stopping the hierarchical segmentation, and instead simply select a number of segments to match results of prior work (this aspect of the segmentation algorithm is orthogonal from our work on the partition function, and thus we have not addressed it in our system).

7.3 Surface Correspondence

Finding correspondences between points on two different surfaces is a fundamental problem in computer graphics – it is an underlying problem in surface matching, morphing, completion, symmetrization, and modeling.

While there are many algorithms for finding a dense inter-surface mapping when given an initial coarse set of point correspondences (e.g. [Schreiner et al. 2004]), it is still very difficult to establish the initial correspondences automatically, even for rigid-body transformations. The problem is to derive an alignment error function that heavily penalizes mismatches of semantic features. Typical methods based on RMSD and local shape descriptors are not always effective [Audette et al. 2000], and thus most systems rely upon human-provided markers (e.g., [Allen et al. 2003]).

In this section, we propose that the partition function can be used to help establish a coarse set of correspondences automatically between two surfaces. Our intuition is based on the results shown in Sections 6 – the partition function is high in concave seams where large parts connect. Since those seams are often a consistent feature across a set of deformations and/or within a class of objects (Figure 9), we hypothesize that they provide a useful cue for establishing point correspondences.

To investigate this hypothesis, we have implemented a simple adaptation of the RMSD error measure for surface alignment. Rather than measuring the sum of squared distances between closest points for all points on two surfaces, we computed a weighted sum of the squared distances between closest points on edges with high partition function values. That is, given two surfaces, we compute their partition functions, sample points from edges with values above a threshold in proportion to their lengths and partition function, and then use ICP with multiple restarts to find the similarity transformation that minimizes the RMSD of the sampled point sets. This procedure effectively sets weights on the original RMSD to strongly favor correspondences between points on the most consistent cuts.

Figure 15 shows an example of how this simple scheme for aligning cuts can help find a coarse alignment for one model of a horse (red) with six others in different poses (green). The top row shows alignments produced with our cut-weighted alignment scheme, while the bottom row shows the results produced without it (i.e., sampling points on all edges uniformly). Note that head and tail are flipped for three of the examples in the bottom row (they are marked with a black 'X'), while all examples in the top row find good correspondences. In this case, the strong cuts along the neck, across the face, and at the base of the tail provide the main cues that produce semantically correct coarse alignments.

7.4 Deformation

Methods for skeleton-free deformation of a mesh have recently been developed (for example, [Lipman et al. 2005; Botsch et al. 2006] among others). They often allow a user to select a region of influence and a handle, and then move or re-orient the handle while the rest of the mesh deforms in a way that locally preserves surface shape while meeting the user-specified handle constraints. Such methods generally spread deformation error uniformly over a surface, which gives the surface a "rubbery" feel, sometimes creating undesirable deformations (see Figures 16a and 16c).

To address this issue, [Popa et al. 2006] described a "material-aware" deformation framework, where the stiffness of deformation is determined by material properties of the mesh. However, a user had to provide stiffness properties manually (or they had to be learned from example deformations). Here, we use our partition function to provide stiffness properties automatically. Our intuition is that edges of the mesh with high partition function values appear along seams of large parts, and thus should be very flexible, while edges having low partition function values should remain stiff.

Our implementation is based on the deformation framework of [Lipman et al. 2005]. Their framework reconstructs a mesh from local coordinates in two steps: first, the Frenet frames of vertices are solved for to maintain the changes in frames between adjacent vertices, and second, the positions of the vertices are found. We modify the first step so that edges neighboring *high* partition function edges are *less* obligated to preserve frame changes from the original model. To extend the partition function to neighboring edges, if $p(i)$ is the partition function on edge i, we form $p'(i)$ as $p'(i) = \max_{j \in N(i)} p(j)$ (where $N(i)$ is the set of edges adjacent to edge i). We then invert the partition function as: $w(i) = \frac{\alpha}{\alpha + p'(i)}$. These $w(i)$ are the weights we use in the linear system that reconstructs frames to determine how closely the change in frames across edge i must be preserved from the original model. We choose $\alpha = .5$, so that an edge bordering a definite partition has a weight of 1/3, and an edge bordering no partition has a weight of 1.

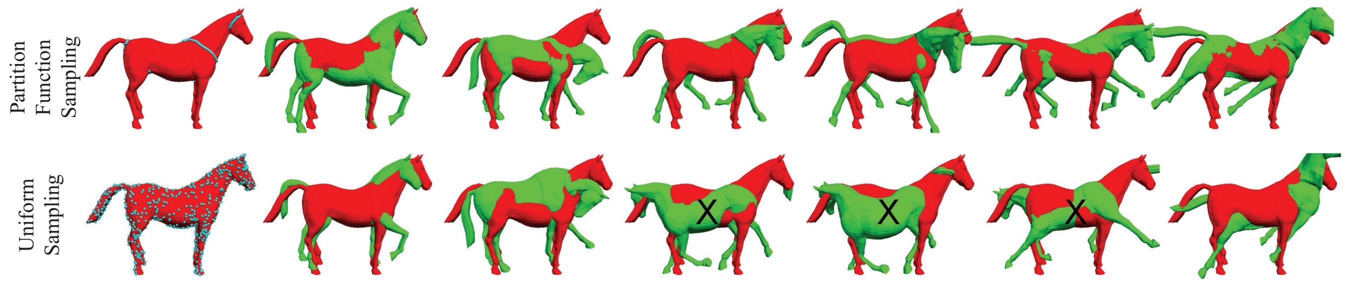


Figure 15: Using the partition function for registration. Alignments based on RMSD between points sampled (left image) according to the partition function (top row) provide better point correspondences than ones based on points sampled uniformly (bottom row).

Figure 16 compares deformations created with (16b and 16d) and without (16a and 16c) weights determined by the partition function. Note that deformations with weights from the partition function preserve the rigid shape of the head and leg of the cheetah, spreading the deformation into the likely segment junctions favored by the partition function. In contrast, the deformations with uniform weights warp the head and leg of the cheetah in an unnatural-looking way.

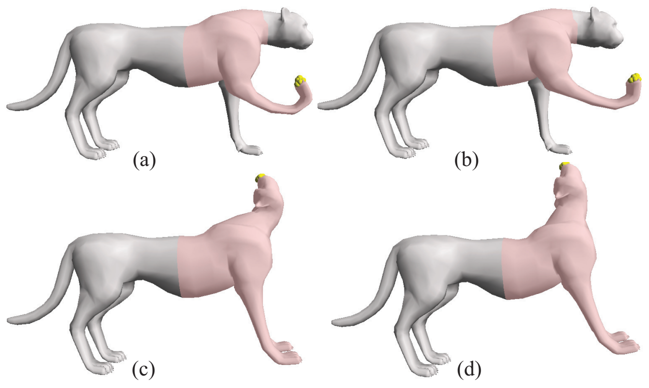


Figure 16: Spreading deformation error uniformly can lead to unnatural deformations (a), (c), whereas allowing more deformation near edges of high partition value leads to more natural, segment-like deformations (b), (d) The deformations were made by setting a region of influence (pink) and adjusting the orientation of handles (yellow spheres).

8 Conclusion

The main contribution of this paper is the idea that randomized cuts can be used for 3D shape analysis. This idea is an instance of a broader class of techniques where randomization of discrete processes produce continuous functions. In our case, the combination of multiple randomized mesh segmentations produces a continuous partition function that provides global shape information about where part boundaries are likely to be on a mesh. This information is stable across several common mesh perturbations, and thus we find it useful in mesh visualization and processing applications.

While the initial results are promising, there are many limitations, which suggest topics for further study. First, our MinCut algorithm produces unstable results for symmetric objects (such as the chest of the Armadillo in Figure 9), as it favors one of multiple cuts of similar costs. Perhaps this problem could be ameliorated by randomizing graph edge weights, but further investigation is required. Second, our study considers a limited set of methods for randomizing segmentation algorithms. For example, it might be useful to randomize algorithm selection, numbers of segments, scales, and other parameters in ways that we did not investigate. Third, we have considered only scalar representations of the partition function. Perhaps it could be extended to multiple dimensions by encoding for each edge the value of the partition function for different scales, numbers of parts, and other parameters that affect segmentations. This could provide a feature vector for every edge that could be useful for visualization or shape matching. Fourth, our study of applications is a small first step. For example, we believe that the partition function could be leveraged more effectively for segmentation, and perhaps it could be used for chokepoint analysis, saliency analysis, feature-preserving smoothing, skeleton embedding, grasp planning, feature detection, and other applications in computer graphics.

9 Acknowledgments

We would like to thank several people who provided suggestions, data, code, feedback, and encouragement for this project, especially Szymon Rusinkiewicz, Philip Shilane, Lior Shapira, and Marco Attene. We also thank Aim@Shape, Stanford, Cyberware, Lior Shapira, Marco Attene, Daniela Giorgi, Ayellet Tal, and others for providing 3D surface models, and we acknowedge NSF (CNFS-0406415, IIS-0612231, and CCF-0702672) and Google for providing funding to support this project.

References

AGATHOS, A., PRATIKAKIS, I., PERANTONIS, S., SAPIDIS, N., AND AZARIADIS, P. 2007. 3D mesh segmentation methodologies for CAD applications. *Computer-Aided Design & Applications 4*, 6, 827–841.

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM Press, New York, NY, USA, 587–594.

ANTINI, G., BERRETTI, S., DEL BIMBO, A., AND PALA, P. 2005. 3D mesh partitioning for retrieval by parts applications. In *Multimedia and Expo*.

ATTENE, M., KATZ, S., MORTARA, M., PATANE, G., AND A ND A. TAL, M. S. 2006. Mesh segmentation - a comparative study. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*, IEEE Computer Society, Washington, DC, USA, 7.

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3, 181–193.

ATTENE, M., ROBBIANO, F., SPAGNUOLO, M., AND FALCIDIENO, B. 2007. Semantic annotation of 3D surface meshes based on feature characterization. *Lecture Notes in Computer Science 4816*, 126–139.

AUDETTE, M. A., FERRIE, F. P., AND PETERS, T. M. 2000. An algorithmic overview of surface registration techniques for medical imaging. *Medical Image Analysis 4*, 3, 201–217.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: coupled prisms for intuitive surface modeling. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 11–20.

CHAZELLE, B., DOBKIN, D., SHOURHURA, N., AND TAL, A. 1997. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry: Theory and Applications 7*, 4-5, 327–342.

EDMONDS, J., AND KARP, R. M. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM 19*, 2.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. *ACM Transactions on Graphics (Siggraph 2004)* (Aug.).

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 1997*, Computer Graphics Proceedings, Annual Conference Series, 209–216.

GARLAND, M., WILLMOTT, A., AND HECKBERT, P. 2001. Hierarchical face clustering on polygonal surfaces. In *ACM Symposium on Interactive 3D Graphics*, 49–58.

GDALYAHU, Y., WEINSHALL, D., AND WERMAN, M. 2001. Self-organization in vision: Stochastic clustering for image segmentation, perceptual grouping, and image database organization. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 10, 1053–1074.

GELFAND, N., AND GUIBAS, L. 2004. Shape segmentation using local slippage analysis. In *Symposium on Geometry Processing*, 214–223.

GIORGI, D., BIASOTTI, S., AND PARABOSCHI, L. 2007. SHape REtrieval Contest 2007: Watertight models track. In *SHREC competition*.

HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. 2001. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 203–212.

INOUE, K., TAKAYUKI, I., ATSUSHI, Y., TOMOTAKE, F., AND KENJI, S. 2001. Face clustering of a large-scale cad model for surface mesh generation. *Computer-Aided Design 33*, 251–261.

KARGER, D. R., AND STEIN, C. 1996. A new approach to the minimum cut problem. *Journal of the ACM 43*, 4, 601–640.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (TOG) 22*, 3, 954–961.

KATZ, S., LEIFMAN, G., AND TAL, A. 2005. Mesh segmentation using feature point and core extraction. *The Visual Computer (Pacific Graphics) 21*, 8-10 (October), 649–658.

LAI, Y., HU, S., MARTIN, R., AND ROSIN, P. 2008. Fast mesh segmentation using random walks. In *ACM Symposium on Solid and Physical Modeling*.

LEE, C. H., VARSHNEY, A., AND JACOBS, D. W. 2005. Mesh saliency. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, 659–666.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H. 2005. Mesh scissoring with minima rule and part salience. *Computer-Aided Geometric Design 22*, 5, 444–465.

LI, X., WOON, T. W., TAN, T. S., AND HUANG, Z. 2001. Decomposing polygon meshes for interactive applications. In *Proc. Symposium on Interactive 3D Graphics*, ACM, 35–42.

LIN, H., LIAO, H., AND LIN, J. 2004. Visual salience-guided mesh decomposition. In *IEEE Int. Workshop on Multimedia Signal Processing*, 331–334.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3, 479–487.

LIU, R., AND ZHANG, H. 2004. Segmentation of 3d meshes through spectral clustering. In *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*.

MANGAN, A., AND WHITAKER, R. 1999. Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics 5*, 4, 308–321.

MILLER, G. 1994. Efficient algorithms for local and global accessibility shading. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 319–326.

MORTARA, M., PATANE, G., SPAGNUOLO, M., FALCIDIENO, B., AND ROSSIGNAC, J. 2003. Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. *Algorithmica 38*, 1, 227–248.

MORTARA, M., PATAN, G., SPAGNUOLO, M., FALCIDIENO, B., AND ROSSIGNAC, J. 2004. Plumber: a method for a multi-scale decomposition of 3D shapes into tubular primitives and bodies. In *ACM Symposium on Solid Modeling and Applications*.

POPA, T., JULIUS, D., AND SHEFFER, A. 2006. Material-aware mesh deformations. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, IEEE Computer Society, Washington, DC, USA, 22.

SCHREINER, J., ASIRVATHAM, A., PRAUN, E., AND HOPPE, H. 2004. Inter-surface mapping. *ACM Transactions on Graphics (Proc. SIGGRAPH 2004) 23*, 3, 870–877.

SHAMIR, A. 2006. Segmentation and shape extraction of 3d boundary meshes (state-of-the-art report). In *Eurographics*, 137–149.

SHAPIRA, L., SHAMIR, A., AND COHEN-OR, D. 2008. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Vis. Comput. 24*, 4, 249–259.

SHI, J., AND MALIK, J. 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8, 888–905.

SHLAFMAN, S., TAL, A., AND KATZ, S. 2002. Metamorphosis of polyhedral surfaces using decomposition. In *Eurographics 2002*, 219–228.

WU, K., AND LEVINE, M. 1997. 3D part segmetnation using simulationed electrical charge distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 19*, 11, 1223–1235.

YU, S. X., AND SHI, J. 2003. Multiclass spectral clustering. In *International Conference on Computer Vision*, 313–319.

ZHANG, H., VAN KAICK, O., AND DYER, R. 2007. Spectral methods for mesh processing and analysis. In *Eurographics State of the Art Report*.

ZHUKOV, S., INOES, A., AND KRONIN, G. 1998. An ambient light illumination model. In *Rendering Techniques*, 45–56.

ZUCKERBERGER, E., TAL, A., AND SHLAFMAN, S. 2002. Polyhedral surface decomposition with applications. *Computers & Graphics 26*, 5, 733–743.