# The Bottleneck Geodesic: Computing Pixel Affinity

Ido Omer Michael Werman
The Hebrew University of Jerusalem, Israel
Ross Building, Givat Ram Campus
91904 Jerusalem, Israel
{idom,werman}@cs.huji.ac.il

## Abstract

*A meaningful affinity measure between pixels is essential for many computer vision and image processing applications. We propose an algorithm that works in the features' histogram to compute image specific affinity measures. We use the observation that clusters in the feature space are typically smooth, and search for a path in the feature space between feature points that is both short and dense. Failing to find such a path indicates that the points are separated by a bottleneck in the histogram and therefore belong to different clusters. We call this new affinity measure the "Bottleneck Geodesic". Empirically we demonstrate the superior results achieved by using our affinities as opposed to those using the widely used Euclidean metric, traditional geodesics and the simple bottleneck.*

## 1. Introduction

Calculating an affinity measure between pixels is a fundamental problem in low level vision. Many computer vision and image processing algorithms rely on these measures. These affinities are either measured between different pixels of the same image in applications such as segmentation, edge detection and noise reduction, or between pixels from neighboring frames in motion segmentation or tracking.

In practice, most applications calculate pixels affinity as a simple function of the Euclidean distance between the pixels' features (usually $e^{\frac{-d^2}{\sigma^2}}$ where $d$ is the Euclidean distance in some feature space and $\sigma$ is a normalization factor). Common feature spaces are one-dimensional gray scales, two or three dimensional color spaces and higher dimensional ($\sim 50D$) texture feature spaces. Different researchers suggest using different feature spaces for achieving optimal results in various applications, but no particular feature space is considered optimal by the whole community (a survey of the properties of different color spaces for image segmentation can be found in [3], while [16] provides a basic survey of different texture features). Although this approach is efficient and simple, it does not utilize general image properties (class properties) or image specific characteristics. Approaches to the problem that try to use these properties include learning pixels affinity [7], feature space clustering [4, 11] and geodesic distances [14, 6, 8].

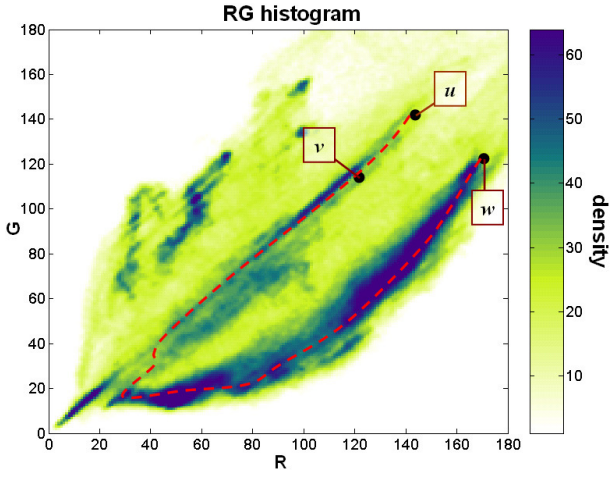



Figure 1. An RG histogram of an image (an RGB histogram, projected upon the RG axis). The histogram reveals that $u$ and $v$ likely belong to the same cluster while $u$ and $w$ do not, although the Euclidean distance between the points $u$ and $v$ is similar to the distance between the points $u$ and $w$. Using [robust] path based similarity still won't help since there is a very long, but dense path between $u$ and $w$ (shown in dashed red line). Nevertheless there is no short and dense path between $u$ and $w$ therefore both sources of information should be used together for providing a meaningful affinity

Model based algorithms for affinity (or distance) calculation that use geodesics were previously suggested in computer vision and several other fields of computer science. These works usually store the data in a graph where each data point (feature points) is represented by a vertex, and neighboring data points are connected using an edge.

An example can be found in Magnus's distance calculations algorithm [14]. His algorithm calculates model based geodesic distances according to the data points (using a graph - shortest paths algorithm) but doesn't use the density information in the feature space. He demonstrates his algorithm only for simple synthetic examples. In their work "Path Based Clustering", Fischer et al suggested a different approach to the problem [6]. Their observation is that many real world problems deal with non compact clusters, and completely eliminate the path length from their minimization term. Instead of calculating the similarity between feature points according to the length of the shortest path between the points, they consider two points as similar if there exists a path without an edge with large cost between the points (a max-min algorithm). The similarity $s_{ij}$ between vertices $i$ and $j$ is expressed by the following term

$$s_{ij} = \max_{p \in P_{ij}} \left\{ \min_{1 \le h < |p|} s_{p[h]p[h+1]} \right\}$$

where $s_{ij}$ is the similarity between nodes $i$ and $j$, $P_{ij}$ denotes the set of all paths between $i$ and $j$, $p[h]$ is a node along such path and $s_{p[h]p[h+1]}$ is the similarity score of an edge along the path (edge length). Their minimization term can be looked at as an approximation of the feature space density. Recently, Chang et. al. suggested adding robustness to the algorithm by multiplying the edge similarity estimator by a density estimator for the edge's vertices [8]. Their algorithm, "Robust Path Based Clustering", is similar to that proposed in [6] and their similarity measure is given by the formula:

$$s_{ij} = \max_{p \in P_{ij}} \left\{ \min_{1 \le h < |p|} w_{p[h]} w_{p[h+1]} s_{p[h]p[h+1]} \right\}$$

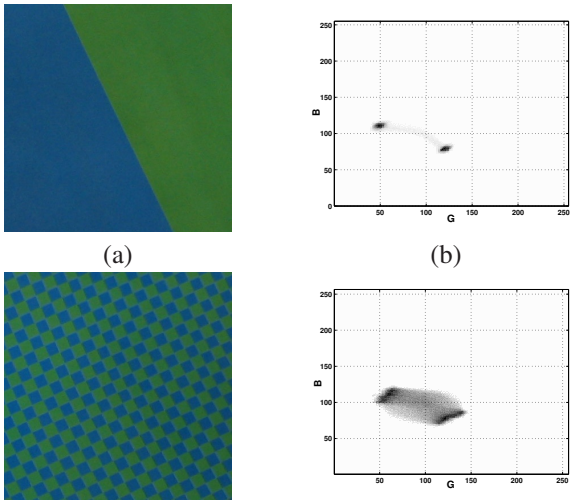
This work adopts the observation that stands at the heart of the [robust] path based clustering and use density information along the path between feature points in the feature space, but unlike these two works, we claim that density information alone is insufficient. Our distances are therefore calculated as a function of the geodsic distance between the feature points and of the minimal density (*Bottleneck*) along this path and is therefore called *Bottleneck Geodesic*. In a previous work [12], we suggested calculating the distance between $RGB$ feature points by deviding the Euclidean distance between the points by the minimal histogram value along the straight line (in the histogram domain) connecting these points (the algorithm works with slight modification for texture as well). This algorithm is simple and efficient but assumes that the feature points form convex (or nearly convex) clusters in feature space. In this work we relax our convexity assumptions and suggest an algorithm that searches for a path in the features graph that is both short and dense.

The motivation for our approach is made clear by looking at the two-dimensional feature histogram in Figure 1. Our main observation is that the histogram provides us with additional knowledge, even though the Euclidean distance between points $u$ and $v$ is equal to the distance between points $u$ and $w$, points $u$ and $v$ likely belong to the same source and should be considered similar, $u$ and $w$ seem to belong to two different sources and should be considered dissimilar. Using [robust] path based similarity does not help in this case, since there is a dense, but very long, path between $u$ and $w$ (shown in a dashed line). There is no path between $u$ and $w$ that is both short and dense.
This work suggests a straightforward approach to affinity calculations which exploits image specific attributes while not explicitly clustering the feature space. We do so by introducing the *Bottleneck Geodesic* - a simple mechanism for estimating the likelihood that two feature points belong to the same cluster in the feature space. We use the histogram density function as a tool for deciding bottlenecks values: a low histogram value along the geodesic between two feature points indicates a narrow bottleneck that decreases our affinity score. Our geodesic measure is therefore calculated as a function of the length of the geodesic where the bottleneck values add extra weight along the path. Given a density function, this approach can be used directly in any given feature space and requires no training stage.

Our algorithm is well adapted for pixels' affinity calculations due to the smooth structure of clusters in the *RGB* histogram of images. This structure is the result of scene properties and the (digital) image acquisition process. We discuss the structure of the clusters in the next section (section 2). Section 3 describes our algorithm and discusses implementation issues. The results are shown in section 4, while section 5 summarizes and suggests possible extensions to this work.

## 2. Histogram Clusters

This section discusses the physical properties that affect cluster structure in an image feature space histogram.
Figure 1 shows a simple image, containing a small number of dominant colors along with a two-dimensional projection of its RGB histogram. The difficulty in modeling the clusters in the histogram domain is evident from this example. The clusters have no particular shape, and methods like the *Gaussian Mixture Model* (GMM) are not suitable for this kind of problem. The large amount of noise makes the task of clustering a difficult one even for this simple scene. Nevertheless, it is obvious that the histogram contains different clusters. For **most** pairs of feature points, the problem of estimating a likelihood measure for the points to belong to the same cluster seems significantly easier than the actual clustering problem. Our algorithm is aimed at utilizing this observation.




(a) (b) (c) (d)

Figure 2. (a) An image of a simple scene containing two objects and its GB histogram (b). (c) An image of a simple scene containing a texture (two $texels$) and its GB histogram (d).

For the sake of simplicity, this discussion refers to three-dimensional color features (*RGB* values). We will address the generalization of the discussion to other feature spaces at the end of this section.

Our algorithm takes advantage of two well studied image properties. The first property is the *piecewise smooth world* assumption which has been used before in many computer vision and image processing applications such as boundary detection [10], image segmentation [17], noise reduction [15] and more. The second property is the image blur due to the optics of the camera and the finite size of the pixel [13].

The first property implies that for two feature points belonging to the same monochromatic object, there is a path of histogram bins connecting the points (in the histogram domain) that is well populated with points from the same monochromatic object. Due to the second property, the same holds for textured objects as well. The justification for the last claim lies in the following fact: While locally, in the image plane, there is no difference between the blurring of edges due to texture and due to boundaries, globally - in the histogram domain there is a big difference between these phenomena. Boundaries between objects are sparse and therefore produce a small number of interpolated values. The line in the histogram between pixels from neighboring objects is therefore sparseely populated. In textured regions the same texture components (texels) are blurred repeatedly. The line between pixels from two different texels (of the same texture) in the histogram is well populated.

Figure 2 demonstrates the difference in the histogram domain between edges due to object boundaries and edges due to texture. In the figure we show real images of a simple scene along with their $GB$ histogram (a projection of their $RGB$ histogram upon the $GB$ axis). There are only a few pixels with interpolated values in figure (a) and the two clusters are well separated in the histogram domain (b). In figure (c) many of the pixels have interpolated values and the region between the two clusters in the histogram domain (d) is densely populated.

(a) (b) (c)

Figure 3. (a) An image (b) the square root of its Euclidean edge map (c) the square root of its edge map as computed according to the bottleneck geodesics (the differences are more easily seen with square root). The values in both maps are normalized to the range of [0..1]. Notice how the bottleneck edge map maintains edges between objects while the edge values inside the sculptures are significantly lowered.

Figure 3 shows an image along with the edge map according to the Euclidean metric (b) and to our bottleneck distance measure (c). Notice how the bottleneck edge map maintains edges between objects while the edge values inside the sculptures are significantly lowered.

Although in the entire section we referred to $RGB$ features, we would like to point out that our main argument, smoothness due to scene and camera properties, is a fundamental property of natural images and is not related to a specific set of features. Our experiments support this observation as we clearly show in the results section.

## 3. Defining & Computing Bottleneck Geodesics

The first step in computing the *bottleneck geodesic* is representing the feature space using a graph. A feature point is represented as a node in the graph. Each node stores its neighborhood's density estimation (a *bottleneck* value). For a feature point $x_i$ and a neighborhood $N_i$ around it, we calculate $x_i$'s density using the formula $\sum_{x_j \in N_i} \frac{1}{\|x_i - x_j\|^2}$. We connect each node in the graph to a fixed number of its nearest neighbors (usually 5) using edges, the edge length is set to the Euclidean distance between the nodes (feature points).

For computing the *bottleneck geodesic* between two graph nodes (two feature points) we need to find a path connecting the nodes in the graph, that minimizes the length of the path (geodesic distance), while avoiding low density (narrow bottleneck) regions. Although finding an optimal

path according to each one of the two requirements is a simple problem and can be computed using a shortest path algorithm, finding a path that is optimal according to both requirements is a difficult problem (NP hard). Using a greedy algorithm (Dijkstra) does not guarantee to find the optimal path.

One possible solution is coupling the density and distance information in the edges themselves. We do so by dividing the edge length by the minimal bottleneck value of the edge end nodes, creating *Local Bottleneck Geodesic* (LBG). This way we get rid of the information stored in the graph nodes and are left with a standard graph (with modified edge weights) for which we calculate the shortest paths using Dijkstra's shortest paths algorithm [5]. Although this guarantees that the path we find is optimal for our minimization term (path length), moving the bottleneck information from the nodes to the edges means that the density information has only a local effect. Instead of dividing the length of the whole path by the bottleneck value, we only divide the length of a single edge by that value.

A different approach is to use Dijkstra's algorithm for finding paths that are optimized according to both the length and global density (bottleneck) terms. Although the algorithm does not guarantee to provide the optimal paths, in practice it yields good results. An example for a situation were the algorithm will not find an optimal path is when the algorithm searches for the shortest path between a dense node $u$ and a sparse node $v$. Let us assume that the path goes through a node $w$ which is denser than $v$. The algorithm might prefer a longer path between $u$ and $w$ to avoid sparse regions, but since $v$ is in itself a sparse node, the algorithm will later pay more for its greedy decisions. In practice, as revealed by our empiric experiments, this is usually not the case and the algorithm computes meaningful affinities that achieve excellent segmentation benchmark results. This is probably mainly due to smoothness of the data, but we believe a more thorough study of the criteria for correctness of the greedy algorithm should be performed and leave it for future work.

There are several ways for coupling the length and (global) density of the path. We have tried two such approaches. In the first approach we define the "price" of the path $p_{uv}$, connecting the nodes $u$ and $v$ as $\frac{p_{uv}.length}{p_{uv}.bottleneck}$ where $p_{uv}.length$ is the length of the path, and $p_{uv}.bottleneck$ is the minimal bottleneck (density) along the path, we will call this approach *Bottleneck Geodesic Ratio* (BGR). In the second approach, *Interpolated Bottleneck Geodesic* (IBG), the price is defined as $\frac{q}{p_{uv}.bottleneck} + (1-q) \cdot p_{uv}.length$ where $q$ is a parameter (choosing $q$ to be in the range of $[\frac{1}{2}, \frac{3}{4}]$ gave good results). We have evaluated the three approaches using the Berkeley segmentation benchmark and as shown in the results section, they all gave good results, although the global bottleneck approach (GBR/IBG) performed better.

Computationally, calculating the bottleneck affinities between one pixel and the rest of the image (single source shortest path) requires $\theta(|V|log(|V|) + |E|)$. Since we only have edges between neighboring bins, $|E| = O(|V|)$ hence the computational cost is $\theta(|V|log(|V|))$ ($|V|$ is bounded from above by the number of image pixels). Computing the affinities between all pairs of pixels requires $\theta(|V|^2 log(|V|))$ operations (instead of $\theta(|V|^2)$ operations for a naive Euclidean computation). We have implemented our algorithm in C++, but our code is far from being optimal. In practice, producing affinity measures in a 11*11 neighberhood around each pixel in the dataset took 15 minutes on average for a 300*200 image (on a Pentium IV, 2.4GH). For comparison, segmenting the images to 9 segments given the affinities took around 5 minutes per image (segmenting the images to a larger number of segments takes more time). Fischer et al [6] suggest in their paper, various ways for speeding the algorithm up including building a multi scale graph, but we didn't implement any of these methods since our software was only developed for a concept demonstration. In our previous work ([12]), when calculating affinities between high dimensional features (texture features) we had to reduce the features dimensionality. Without the dimensionality reductions, the feature space histogram becomes meaningless since the chance for two pixels to be mapped to the same bin approaches zero. For calculating affinity between high dimensional texture features we therefore first projected the features unto their first three principal components using $PCA$ and calculated the affinities between the projected features. Our results show that in spite of the dimensionality reduction, our affinities using the projected features perform better than the Euclidean metric in the full dimensional space. Although the graph representation frees us from having to reduce the features dimensionality, we still projected the texture features to 3D dimensions. The main reason for doing so was computational efficiency, not projecting the texture features leads to very large graphs where the running time grows dramatically due to memory management (disk access).

## 4. Results

A natural way to evaluate a pixel affinity measure is by using these measures in a segmentation algorithm and evaluate the segmentation performance. We chose the *Ncut* algorithm for our experiments since it is a well known image segmentation algorithm that is easily adopted to use with various affinity measures. Pixels affinities were computed in an 9*9 neighborhoods around each pixel and the images were segmented automatically using the *Ncut* algorithm

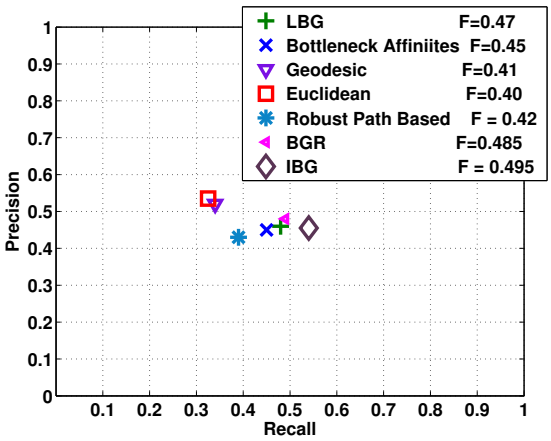



Figure 4. Precision/Recall and $F-measure$ scores for the $Ncut$ algorithm using color features and various affinity measures

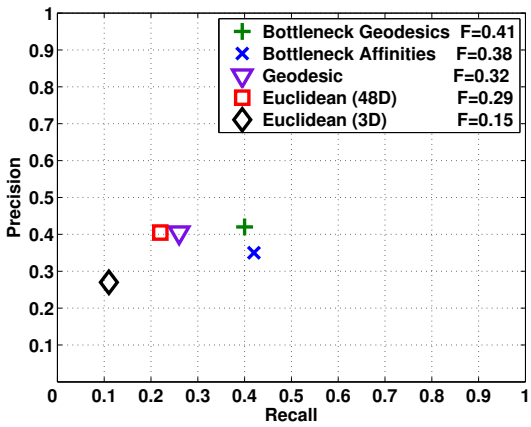



Figure 5. Precision/Recall and $F-measure$ scores for the $Ncut$ algorithm using texture features and various affinity measures

into 9 segments (the segments are not necessarily continuous). For providing a quantitative evaluating of the segmentation results we used the Berkeley segmentation benchmark mechanism [1]. The Berkeley segmentation dataset contains 100 test images. For each of these images several human segmentations are provided (usually 5). These manual segmentations are considered ground truth for evaluating the segmentation results. The evaluation is performed by computing **precision** and **recall**. Precision is the probability that a boundary pixel produced by the algorithm is a true boundary pixel, while recall is the probability that a true boundary pixel is detected. These measures are combined to form an **F-measure** which is the harmonic mean of the precision and recall measures. We evaluated our different *Bottleneck Geodesics* affinities calculations for $RGB$ color features and provide both their benchmark results and a few qualitative results. We also provide a limited evaluation of our algorithm for calculating affinities between texture features. The texture features used are the Leung-Malik filter bank [9] (a total of 48 filters). The filter banks code was obtained from [2]. We compared the segmentation results achieved using *Bottleneck Geodesics* with those achieved using *Bottleneck Affinities* ([12]), simple geodesic distances ([14]), path based affinities ([8]) and Euclidean distances. When using texture features we provide benchmark results for both the full dimensional Euclidean space (48D) and for the projected Euclidean space (3D). We demonstrate how the *Bottleneck Affinities* achieves results that are superior to those achieved by its competitors for both color and texture features. Figure 4 shows the average precision and recall achieved by the *Ncut* algorithm using the various affinities and color features. The average precision and recall achieved using texture features is shown in Figure 5. A few qualitative examples (using color features) are provided in Figure 6.

## 5. Discussion

We introduced the *bottleneck geodesic*, a novel image specific approach for calculating similarity measures between pixel features. Our algorithm decreases the affinity (relative to the Euclidean or geodesic distances) between two feature points when it estimates that they belong to two different clusters, while increasing their affinity when estimating they belong to the same cluster. We do so without explicitly clustering the data and with only weak assumptions on the structure of these clusters. Unlike other algorithms that use density information, our algorithm not relies on density alone, rather it combines it with distance (path length) information. We suggested three methods for combining distance and density information. The density information can be used for adjusting the length of the edges in the features graph (dividing the edge length by the minimal density of its two end nodes). This way, all the information is stored in the edges and any shortest paths algorithm is guaranteed to provide correct results. In the two other methods, length information is stored in the graph edges while density information (bottleneck) is stored in the nodes. In these methods we search for a path that is, as short as possible while avoiding sparsely populated regions (narrow bottlenecks) whenever possible. We have suggested two different minimization terms for this approach, $\frac{p_{uv}.length}{p_{uv}.bottleneck}$ and $\frac{q}{p_{uv}.bottleneck}+(1-q)\cdot p_{uv}.length$ where $p_{uv}.length$ is the length of the path, and $p_{uv}.bottleneck$ is the minimal bottleneck along the path ($q$ is a parameter). Although there is no algorithm we know of that is guaranteed to efficiently finds the optimal path according to these terms, in practice, running Dijkstra with these terms gave good results. Our algorithm requires no learning stage. It can be easily applied for calculating distances or affinities between feature vectors in any feature space given a density function on the features. Our implementation uses a histogram as

a density function, but the algorithm is not limited to any particular such function. Although computationally, the algorithm is more time consuming than calculating Euclidean distances, our non optimized implementation achieved running time that is comparable with that of the segmentation algorithm we used. We believe that a better implementation will achieve running times that are acceptable for many applications.

The information in the histogram has not received enough attention and we intend to continue to exploit this information for various computer vision tasks. Our algorithm fills an important gap between two different approaches, one that uses distance information alone (Euclidean distance and traditional geodesic distance) and the other that relies only on density information (path based similarity). We demonstrated that better benchmark results and more visually appealing segmentations where achieved by combining these two cues.

We further intend to apply this method to the analysis of different kinds of data.

## References


[1] http://www.cs.berkeley.edu/projects/vision/grouping. The Berkeley Segmentation Dataset and Benchmark.

[2] http://www.robots.ox.ac.uk/˜vgg. Visual Geometry Group, University of Oxford.

[3] H.-D. Cheng, X. Jiang, Y. Sun, and J. Wang. Color image segmentation: advances and prospects. *Pattern Recognition*, 34(12):2259–2281, 2001.

[4] D. Comaniciu and P. Meer. Robust analysis of feature spaces: color image segmentation. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition*, page 750, Washington, DC, USA, 1997. IEEE Computer Society.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] B. Fischer, T. Zöller, and J. M. Buhmann. Path based pairwise data clustering with application to texture segmentation. *Lecture Notes in Computer Science*, 2134:235–250, 2001.

[7] C. Fowlkes, D. Martin, and J. Malik. Learning affinity functions for image segmentation: Combining patch-based and gradient-based approaches. *Computer Vision and Pattern Recognition*, June 2003.

[8] D. Y. H. Chang. Robust path-based spectral clustering with application to image segmentation. In *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV05)*, volume I, pages 278–285, Octorber 2005.

[9] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *Int. J. Comput. Vision*, 43(1):29–44, 2001.

[10] D. Mumford and J. Shah. Boundary detection by minimizing functionals, I. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 22–26, 1985.

[11] I. Omer and M. Werman. Color lines: Image specific color representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume II, pages 946–953. IEEE, June 2004.

[12] I. Omer and M. Werman. Image specific feature similarities. In *Proceedings of the European Conference on Computer Vision (ECCV06)*, 2006.

[13] S. Park and R. Schowengerdt. Image sampling, reconstruction, and the effect of sample scene phasing. *Applied Optics*, 21:3142–3151, September 1982.

[14] M. Rattray. A model-based distance for clustering. In *IJCNN (4)*, pages 13–16, 2000.

[15] V. Spokoiny. Estimation of a function with discontinuities via local polynomial fit with an adaptive window choice. *Annals of Statistics*, pages 1356–1378, 1998.

[16] M. Tuceryan and A. K. Jain. Texture analysis. pages 235–276, 1993.

[17] L. A. Vese and T. F. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *International Journal of Computer Vision*, 50(3):271–293, 2002.

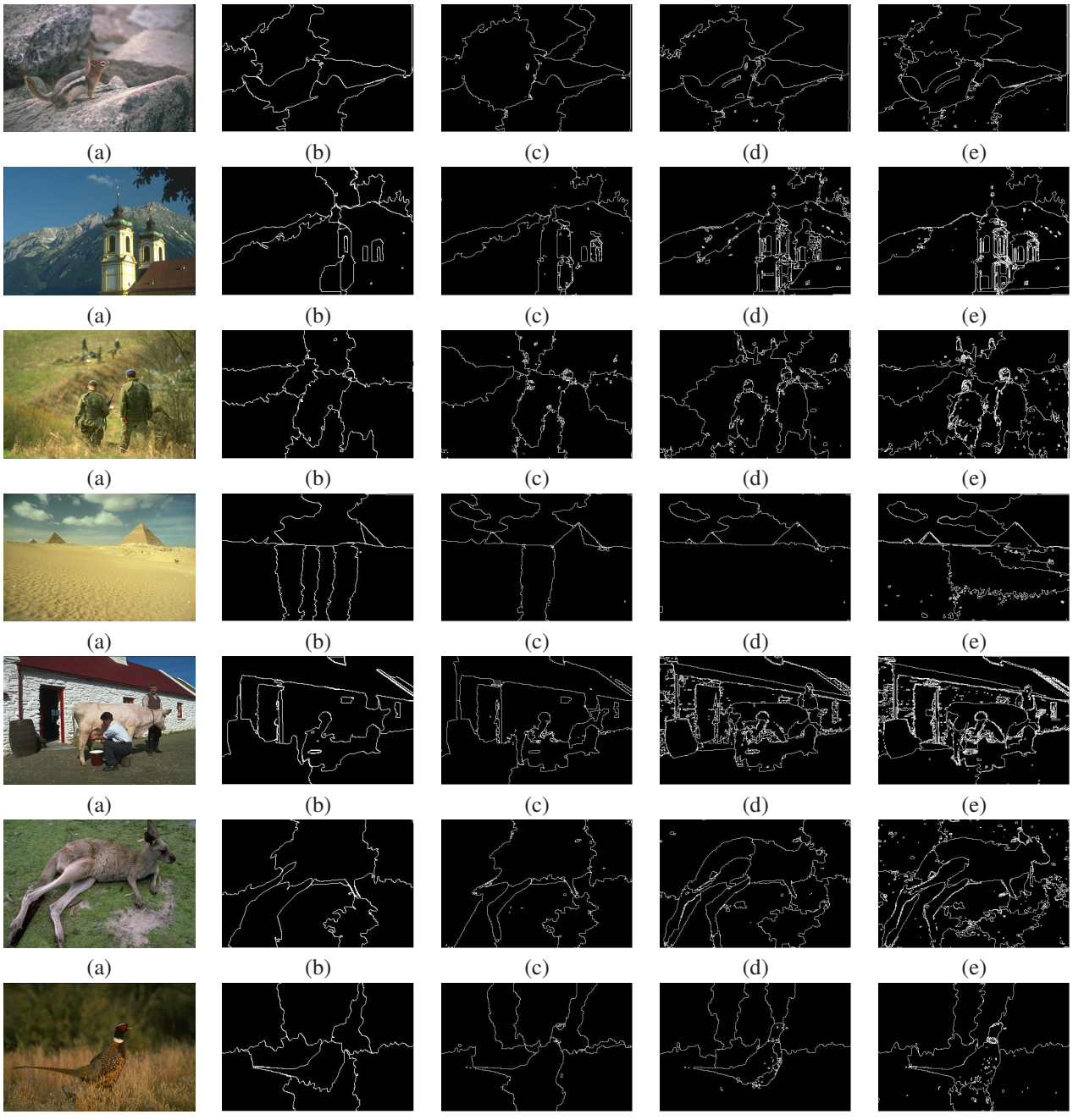



Figure 6. Segmentation to 9 segments using the $Ncut$ algorithm with color features and various affinities: (a) Input image (b) Euclidean affinities (c) Geodesics (d) Local Bottleneck Geodesic (LBG), and (e) Global Bottleneck Geodesics (IBG)