158 D. L. Page, A. F. Koschan, and M. A. Abidi, "Perception-based 3D Triangle Mesh Segmentation Using Fast Marching Watersheds," Proc. Intl. Conf. on Computer Vision and Pattern Recognition, Vol. II, pp. 27-32, June 2003.

# Perception-based 3D Triangle Mesh Segmentation Using Fast Marching Watersheds

D. L. Page A. F. Koschan M. A. Abidi
*Imaging, Robotics, and Intelligent Systems Laboratory*
*University of Tennessee, Knoxville, Tennessee 37996-2100*
*davidpage@ieee.org*

## Abstract

*In this paper, we describe an algorithm called Fast Marching Watersheds that segments a triangle mesh into visual parts. This computer vision algorithm leverages a human vision theory known as the minima rule. Our implementation computes the principal curvatures and principal directions at each vertex of a mesh, and then our hill-climbing watershed algorithm identifies regions bounded by contours of negative curvature minima. These regions fit the definition of visual parts according to the minima rule. We present evaluation analysis and experimental results for the proposed algorithm.*

## 1. Introduction

A 3D triangle mesh is simply a collection of triangles and vertices that approximate a 3D surface. Although this representation is useful for visualization, the low-level description is often inadequate for computer vision tasks such as object recognition, scene understanding, and feature modeling. Higher-level descriptions are necessary, and one way to impose such descriptions is through mesh segmentation.

*Mesh segmentation*, like the more common image segmentation [13], refers to the partitioning of a mesh into more visually meaningful regions, or parts. Although an abundance of literature is available for range image segmentation [3], mesh segmentation has only recently become of interest. From the literature, we suggest two algorithms represent the state of the art: Mangan and Whitaker [7] and Wu and Levine [17]. Other works include [16, 2, 12, 18, 9]. Mangan and Whitaker coin the term, mesh segmentation, and propose a top-down bobsledding implementation of watershed segmentation from image processing. With a different approach, Wu and Levine propose a novel algorithm based on the simulated distribution of electrical charge across the surface of a mesh. They loosely base their algorithm on a human vision theory known as the minima rule [4].

The *minima rule* is an elegant theory that defines a framework for how human perception might decompose an object into its constituent parts. This rule defines part boundaries along lines of negative minima curvature. Based on this theory, [17] leads to meaningful segmentations, but their algorithm, and specifically their curvature estimation, is not true to the minima rule. Wu and Levine acknowledge this limitation in their work and list necessary assumptions such as genus-one topology, no dents on the surface of the object, and the object must have at least one part boundary. None of these assumptions are necessary for our algorithm. Also, we note a difference in computational complexity. Wu and Levine solve a set of linear equations with an iterative conjugate gradient method. With our curvature [15], we have a direct closed form technique. Mangan and Whitaker [7], on the other hand, propose a very robust segmentation strategy, but their method yields an *ad hoc* segmentation without support from a theory such as the minima rule. Building on these methods, we seek an algorithm that strictly adheres to the minima rule and yet leverages the robustness of a watershed segmentation.

As an implementation of the minima rule and as our contribution to the state of the art, we describe in this paper a novel algorithm called *Fast Marching Watersheds*. We highlight the contributions of our algorithm as follows:

- creation of a robust hill climbing algorithm for watershed segmentation on a triangle mesh data structure,
- definition of a directional height map appropriate for the minima rule using local principal curvatures, and
- application of morphological operations to improve the initial marker set for the watershed algorithm.

We investigate these contributions in the remainder of this paper. We begin in Sec. 2 with the theoretical formulation of our algorithm. Then, in Sec. 3, we present experimental results. Finally, we conclude in Sec. 4.

1063-6919/03 $17.00 2003 IEEE

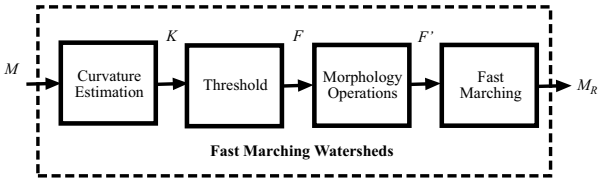



**Figure 1. Block diagram of Fast Marching Watersheds.**

## 2. Theory

Our Fast Marching Watersheds algorithm—both the name and the algorithm—derive inspiration from Fast Marching Methods [6] for computing shortest geodesic paths on a mesh. Kimmel and Sethian's algorithm employs a heap data structure to control the geodesic "walk" across the surface of the mesh. For our watershed algorithm, we use a similar heap structure to control the "flooding" across the mesh. Unlike Kimmel and Sethian, however, our heap keys on local height values, and not cumulative geodesic distances, and it tracks the progression of regions, and not geodesic paths. These differences mark our contribution of extending Kimmel and Sethian's algorithm into the different application of mesh segmentation and the formulation of a flooding-based watershed algorithm. A block diagram of the algorithm appears in Fig. 1.

The input to the algorithm is a triangle mesh $M$. The first three blocks in the diagram create the marker set $F'$ on $M$ to initialize the watershed algorithm. Then, the last block grows the marker regions into the final watershed catchment basins $M_r$ that cover the entire mesh.

$$M = \bigcup_{r=0}^{R-1} M_r = M_R \tag{1}$$

such that $M_r \cap M_s = \emptyset$ when $r \neq s$. Note that $M_R$ simply denotes the segmented version of $M$ into $R$ regions. Fig. 2 shows the progression of the algorithm. We now look in depth at each of the blocks in the following sections.

### 2.1. Marker Set

The front three blocks of Fig. 1 establish the marker set $F'$. The first block computes the curvature set $K$ for the vertices of the surface $M$. The next block thresholds the regions $F$ of positive curvature since the minima rule defines contours of negative curvature as visual part boundaries. The third block uses mathematical morphology to clean up the holes and gaps in $F$ to form $F'$.

The primary difference between our proposed algorithm and that of [7] and [17] is the estimation of surface curvature. Both these previous methods compute a *pseudo* curvature measure and do not directly compute principal curvatures. Principal curvatures are the fundamental definition of

(a) (b) (c)

**Figure 2. This example shows the progression of Fast Marching Watersheds for a coffee mug. (a) An initial threshold of curvature. (b) Morphology operators clean the marker set. (c) Final segmented regions after fast marching.**

curvature for a surface [1] and are essential to the definition of the minima rule [4]. Thus, we use [15] to estimate the principal curvatures, $\kappa_1$ and $\kappa_2$, and the associated principal directions, $T_1$ and $T_2$, for each vertex $v$ of $M$.

In the second block, we then threshold each minimum principal curvature $\kappa_2$ throughout the mesh. This block requires the user to select a threshold value. The minima rule suggests that $t > 0$ for $F$ is the natural choice, but implementation results suggest that a slight negative offset from zero improves results. The question is how far to offset. We have defined this offset as a user parameter for our algorithm. Consider the set of vertices $V^-$ with negative $\kappa_2$ values such that

$$V^- = \{\, v_i \mid \kappa_{2,i} \leq 0 \,\} \tag{2}$$

where $\kappa_{2,i}$ is the minimum principal curvature for vertex $v_i$. We can average across this set and establish our threshold as a percentage $\alpha$ of this average as follows

$$t = \frac{\alpha}{n^-} \sum_{v_i \in V^-} \kappa_{2,i} \tag{3}$$

where $n^-$ is the number of vertices in $V^-$. Thus, the user chooses $\alpha$ instead of the threshold $t$ directly. We suggest that $\alpha = 0.3$ yields good results for most applications. Fig. 2(a) shows the results of thresholding the mug.

The next step after thresholding is to apply mathematical morphology operations to clean up the initial segmentation. Rössl et al. [11] discuss the implementation of morphological operators for the arbitrary connectivity of a triangle mesh using a disk-like structuring element. They base their structuring element on the $k$-ring neighborhood about a vertex where the $k$-ring defines the "radius" of the disk element. So, we apply an opening and a closing operation in sequence to clean our initial threshold set $F$. The user must specify a single parameter $k$, the size of the disk. We choose $k = 1$ for most applications. Thus, we have

$$F' = (F \bullet D_k) \circ D_k \tag{4}$$

where $D_k$ is the disk while $\bullet$ and $\circ$ denote closing and opening operators, respectively. The closing bridges small gaps while opening eliminates small isthmuses. Fig. 2(b) illustrates the effect of morphology for the mug.

### 2.2. Fast Marching Watersheds

To this point, we have specified a marker set $F'$. We now seek an algorithm to grow these catchment basins to segment the entire mesh as in Fig. 2(c). The watershed transformation was originally developed by Meyer and Beucher [8] and converted by Vincent and Soille [16] into a digital algorithm for gray level images. In image processing, the selection of a particular watershed algorithm is not straightforward and often confusing since the literature lacks thoughtful distinctions between algorithm specification and implementation [10]. Roerdink and Meijster [10] attempt to bring some order—at least for image processing—to the situation through a critical review. From this review, we have selected the hill climbing algorithm. Our motivation for choosing this algorithm is that it is the simplest and that its reliance on a local height computation allows us to implement the minima rule. Our implementation appears in Alg. 1.

**input** : Mesh $M$ and an array label relative to $F'$.
**output** : The array label contains no elements that are "unlabeled."

*initialize heap* ;
**foreach** $v_i$ *in* $M$ **do**
  **if** *label* $[v_i]$ **then** `Grow(` $v_i$, *heap*, *label* `)` ;
**end**
**while** *not* `Empty(` heap `)` **do**
  data ← `PopHeap(` *heap* `)` ;
  $v_i$ ← data.vertex ;
  **if** *not label* $[v_i]$ **then**
    label $[v_i]$ ← data.label ;
    `Grow(` $v_i$, *heap*, *label* `)` ;
  **end**
**end**

**Algorithm 1:** Fast Marching Watersheds algorithm for a mesh data structure.

This hill climbing algorithm differs significantly from the bobsledding method of [7] and slightly from the immersion algorithm of [16]. The main advantage as [10] notes is the algorithm simplicity since it does not require merging operations. The heart of our implementation is the `heap` data structure that controls the hill climbing process and the procedure `Grow()` that populates this heap. The key into the heap is local curvature such that positive curvature keys bubble to the top and more negative ones sink to the bottom. The `PopHeap()` function pulls from the top of the heap to begin each stage of the marching process. The data associated with a key specifies to which vertex to march and to what catchment basin it potentially belongs. If we pop a vertex from the heap and it is already labeled, we simply pop the next one.

This climbing heap is similar to the marching heap found in Fast Marching Methods [6]. As with Fast Marching Methods, Fast Marching Watersheds have computational complexity $O(n \log n)$ where $n$ is the number of vertices. This order assumes that we must segment each vertex, but with an initial marker set we typically have anywhere from 70–90% of the vertices already segmented. So, $n$ is usually much less than the actual number of vertices in the mesh. The primary difference however between Fast Marching Methods for geodesics and our Fast Marching Watersheds for segmentation is the function `DirectionalHeight()`. This function is the impetus of the minima rule segmentation.

**input** : Vertex $v_i$, an array label, and a heap heap
**output** : Populates heap with neighbors of $v_i$.

*initialize data* ;
**foreach** $v_j$ *in umbrella neighborhood of* $v_i$ **do**
  **if** *not label* $[v_j]$ **then**
    data.vertex ← $v_j$ ;
    data.label ← label $[v_i]$ ;
    key ← `DirectionalHeight(` $v_i$, $v_j$ `)` ;
    `PushHeap(` *heap, key, data* `)` ;
  **end**
**end**

**Procedure:** `Grow(` *vertex, heap, label* `)` is a procedure that extends the marching boundary.

### 2.3. Directional Curvature Height

For the minima rule and our watershed implementation, we need a height map over the triangle mesh such that we specify the height $h_{ji}$ from vertex $v_j$ to vertex $v_i$, relative to the local curvature between these vertices. Euler's formula for surface curvature serves as a starting point. This equation specifies the normal curvature along a particular direction. For our local curvature height, this direction is from $v_j$ to $v_i$. The tangent unit vector $T_{ji}$ at $v_j$ associated with this direction is as follows:

$$T_{ji} = \frac{\mathbf{T}_j(v_i - v_j)}{\|v_i - v_j\|} \tag{5}$$

where $\mathbf{T}_j$ is a $3 \times 3$ matrix that projects the direction vector $(v_i - v_j)$ into the tangent plane of $v_j$. We define this matrix

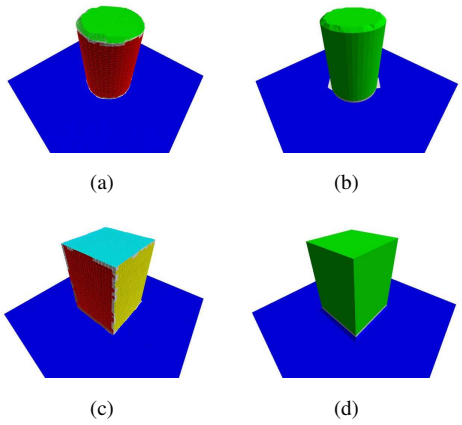



**Figure 3. A comparison of Mangan and Whitaker's algorithm [7] (a,c) and our proposed algorithm (b,d).**

as follows:

$$\mathbf{T}_j = \mathbf{I} - N_j N_j^t \tag{6}$$

where $\mathbf{I}$ is the identity matrix and the vector $N_j$ is the surface normal at $v_j$. The superscript $t$ denotes the transpose and thus leads to an outer product operation. We can compute the normal curvature $\kappa_{ji}$ with Euler's formula as

$$\kappa_{ji} = h_{ji} = \kappa_1 \cos^2(\theta_i) + \kappa_2 \sin^2(\theta_i) \tag{7}$$

where $\kappa_1$ and $\kappa_2$ are the maximum and minimum principal curvatures at $v_j$, respectively. The angle $\theta_i$ is the angle between $T_{ji}$ and the maximum principal direction $T_1$ at vertex $v_j$.

$$\cos\theta_i = T_{ji}^t T_1 \ . \tag{8}$$

Thus, the `DirectionalHeight()` function in the `Grow()` procedure returns $\kappa_{ji}$ as the directional height from $v_j$ to $v_i$. This definition of height allows the watershed algorithm to flow along vertices with similar values of curvature but impedes climbing up negative curvature hills.

## 3. Experimental Results

We now investigate experimental results using the algorithm we have presented. We begin with a qualitative comparison in Fig. 3. This figure demonstrates a simple comparison of our algorithm to [7]. For this comparison, we have not implemented the complete algorithm of [7]. Instead, we have only implemented their curvature estimation method. So, the results in this figure reflect the spirit of their output. As shown in Fig. 3(a), the algorithm in [7] segments the first scene into three parts: the floor, the sides of the cylinder, and the top of the cylinder. By contrast in Fig. 3(b), our proposed algorithm segments the scene into only two parts:

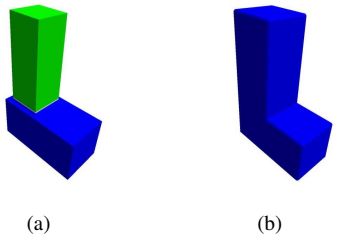



**Figure 4. These examples illustrate a potential drawback of the minima rule.**

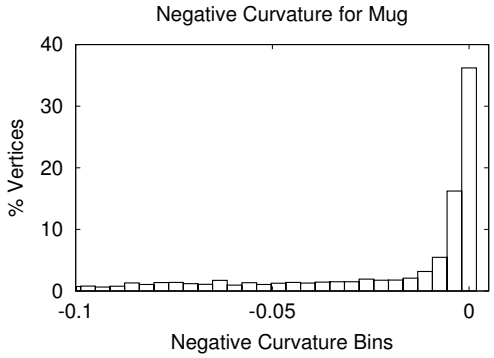



**Figure 5. Distribution of negative principal curvatures for the mug data.**

the cylinder and the floor. Similarly, for Fig. 3(c), their algorithm creates six parts including the floor and each side of the box. Our algorithm in Fig 3(d) again only segments into two parts: the floor and the box. As [4] argues, our algorithm reflects the more likely segmentation that a human observer would choose for the scenes.

Figure 4 demonstrates a potential drawback to our algorithm. In Fig. 4(a), our algorithm segments two clear minima rule parts for the object. The recess of the vertical block relative to the horizontal one forms a closed negative curvature boundary. On the other hand, in Fig. 4(b), no clear minima rule boundary is present. As a result, our algorithm leaves the object unsegmented. This result, however, is not a failure of our algorithm, but rather an ambiguity in the minima rule theory. This ambiguity is discussed in [14].

We next consider the threshold operation and selection of $\alpha$ in Eq. (3). Fig. 5 illustrates why $t > 0$ is not necessarily practical. This figure is a bin plot of the negative principal curvatures for the mug from Fig. 2. The $x$-axis of this plot denotes the bins of negative curvature. The $y$-axis denotes the percentage of vertices with a particular curvature value. From this figure, we see a significant number of vertices exhibit curvature very close to zero. Most likely, these values should be zero, but errors in estimation lead to

(a) (b)

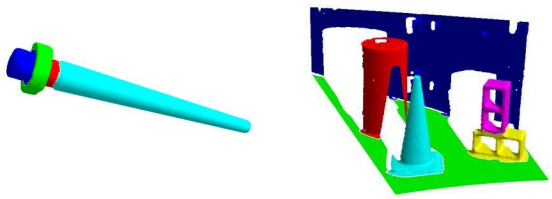
(c) (d)

**Figure 6. Segmentation results for reconstructed meshes from algorithms within our laboratory. (a,c) Hand crank (94K triangles). (b,d) Automotive brake (74K triangles).**

slightly negative values. The parameter $\alpha$ allows us to account for these errors. We have found that the general trend in this plot is typical for the objects we have segmented.

To demonstrate our algorithm, we present results for a series of triangle meshes in Figs. 6, 9, and 7. We use $\alpha = 0.3$ in Eq. (3) and $k = 1$ in Eq. (4) for each of these results. The crank in Fig. 6(c) with seven parts, the disc brake in Fig. 6(d) with two parts, and the scene in Fig. 7(b) with six parts are reconstructions from laser range scans within our laboratory. The watering can in Fig. 9(a) with six parts and chair in Fig. 9(c) with nine parts are reconstructions from the Polhemus Corporation (`www.polhemus.com`). The teapot in Fig. 9(b) with five parts is a reconstruction from [5] (`research.microsoft.com/research/graphics/hoppe/`), and the bunny in Fig. 9(e) with seven parts is from Standford (`graphics.stanford.edu/data/3Dscanrep/`). The bore pin in Fig. 7(a) with four parts is courtesy of the U. S. Army Tank and Automotive Command. The color labels for each object denote the part segmentation results. Again, we emphasize that these segmentations agree with the visual perception of the object parts for most human observers.

Finally, the graph in Fig. 8 shows the timing performance of the proposed algorithm for the mug data set. The implementation for this graph is in Visual C++ on a desktop computer with an Intel Pentium IV at 1.8 GHz with 512 MB memory. To demonstrate the performance, we vary the resolution of the mug mesh with a mesh simplification algorithm. The initial mug consists of $21K$ triangles. We reduce this size in increments of $10\%$ on the $x$-axis. The



(a) (b)

**Figure 7. Segmentation results for reconstructed meshes for (a) a bore pin (75K triangles) and (b) an industrial scene (118K triangles).**

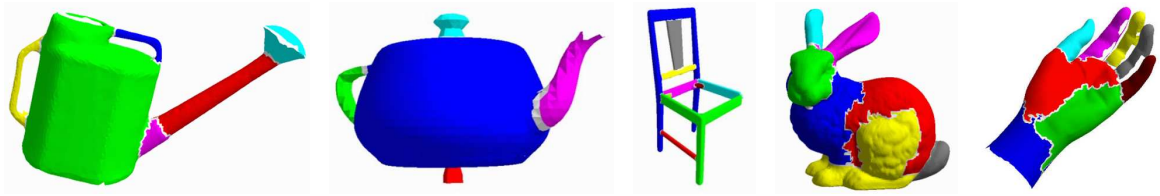
$y$-axis is the processing time required. We plot four curves on the graph. Three of the curves (*Curvature*, *Threshold*, and *Watershed*) represent the computational time for each individual module of the algorithm. The *Threshold* time includes the morphology operations. The *Total* time is the summation of the three modules. The curves appear close to linear except for the jump in the *Threshold* curve from $50\%$ to $60\%$. This jump occurs because below $50\%$ the mug does not require morphology operations to clean up the threshold while above $60\%$ the opening and closing operations are necessary. The jump reflects the performance cost for the morphology operations. We also note that the fast marching module of the algorithm contributes very little to the overall timing while the curvature module contributes the most. Although the $O(n \log n)$ nature of fast marching is inherently fast, the fact that thresholding alone segments typically $70$ to $90\%$ of the mesh improves the performance.

## 4. Conclusions

In this paper, we have introduced a new algorithm for mesh segmentation—Fast Marching Watersheds. We have developed a hill-climbing watershed algorithm to implement the minima rule to achieve our segmentation. We have briefly compared our algorithm to [7] and presented results for a series of mesh reconstructions from range images. These results demonstrate the power of the minima rule to decompose an object into visual parts.

## Acknowledgments

This work is supported by the University Research Program in Robotics under grant DOE-DE-FG02-86NE37968, by the DOD/TACOM/NAC/ARC Program, R01-1344-18, and by FAA/NSSA Program, R01-1344-88/89.



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

**Figure 9. Segmentation results for reconstructed meshes from a variety of sources. (a) Polhemus scans of watering can (16K triangles). (b) Hoppe reconstruction of Utah teapot (6K triangles). (c) Polhemus scans of a chair (53K triangles). (d) Stanford reconstruction of a clay bunny (69K triangles). (e) Polhemus scans of human hand (26K triangles).**

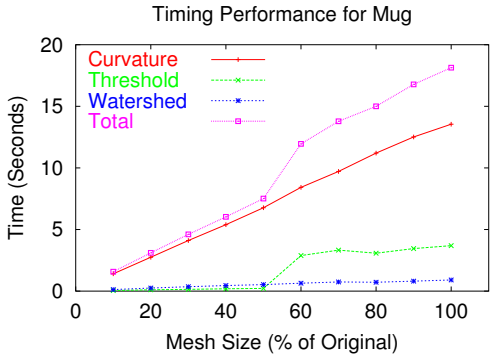



**Figure 8. Timing performance plot of the code modules for the Fast Marching Watersheds as a function of mesh size.**

## References


[1] M. P. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice–Hall, Inc., Englewood Cliffs, NJ, 1976.

[2] B. Falcidieno and M. Spagnuolo. Polyhderal surface decomposition based on curvature analysis. In T. L. Kunii and Y. Shinagawa, editors, *Modern Geometric Computing for Visualization*, pages 57–72. Springer-Verlag, 1992.

[3] P. J. Flynn and A. K. Jain. On reliable curvature estimation. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pages 110–116, 1989.

[4] D. D. Hoffman and W. A. Richards. Parts of recognition. *Cognition*, 18:65–96, 1984.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics Proceedings (SIGGRAPH '92)*, volume 26, pages 71–78, July 1992.

[6] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. In *Proceedings of the National Acadmeny of Sciences*, volume 95, pages 8431–8435, 1998.

[7] A. P. Mangan and R. T. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, Oct.–Dec. 1999.

[8] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communications and Image Representation*, 1:21–45, 1990.

[9] M. E. Rettmann, X. Han, and J. L. Prince. Automated sulcal segmentation using watersheds on the cortical surface. *NeuroImage*, 15(2):329–344, Feb. 2002.

[10] J. B. T. M. Roerdink and A. Meijster. The watershed transform: definitions, algorithms, and parallelization strategies. *Fundamenta Informaticae*, 41:187–228, 2001.

[11] C. Rössl, L. Kobbelt, and H.-P. Seidel. Extraction of feature lines on triangulated surfaces using morphological operators. In *Proceedings of the AAAI Symposium on Smart Graphics*, pages 71–75, 2000.

[12] N. S. Sapidis and P. J. Besl. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Transactions on Graphics*, 14(2):171–200, Apr. 1995.

[13] L. G. Shapiro and G. C. Stockman. *Computer Vision*. Prentice–Hall, Inc., Upper Saddle River, NJ, 2001.

[14] M. Singh, G. D. Seyranian, and D. D. Hoffman. Parsing silhouettes: the short-cut rule. *Perception and Psychophysics*, 61(4):636–660, 1999.

[15] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 902–907, 1995.

[16] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.

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

[18] Y. Yu, A. Ferencz, and J. Malik. Extracting objects from range and radiance images. *IEEE Transactions on Visualization and Computer Graphics*, 7(4):351–364, Oct.–Dec. 2001.