A NOVEL METHOD FOR 3D SURFACE MESH SEGMENTATION

Thitiwan Srinark and Chandra Kambhamettu
Video/Image Modeling and Synthesis (VIMS) Lab.
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716 USA
srinark/chandra@cis.udel.edu

ABSTRACT
We propose a novel method for the problem of 3D surface mesh segmentation. This mesh segmentation method is based on differential geometry and geodesic distance information. In our algorithm, vertices of the same group of surface types are allowed to join a segment. The geodesic distance is estimated using shortest path between connected vertices, which can be computed from well known algorithms such as the Dijkstra algorithm. However, this is inefficient for large meshes. Therefore, in our algorithm, instead of applying the shortest path algorithm to the entire mesh, we locally apply it to mesh partitions so that we can avoid large computational costs. We tested our method with 3D surface meshes of synthetic objects and molecular structures.

KEY WORDS
geometric algorithm, surface segmentation, geodesic distance

1 Introduction

The 3D mesh representation is widely used to represent 3D objects in various applications. For example, in image analysis, it represents reconstructed surfaces of 3D objects from 3D images [1]. Moreover, it is also used to model and visualize complex 3D objects and scenes, which cannot be modeled by any geometric functions. One interesting problem in analysis of surface mesh is surface mesh segmentation. The problem is to cluster vertices of a mesh that are close to each other and have similar curvatures, into the same group. Surface mesh segmentation is applied in many important applications, e.g., feature detection, subpart analysis, model fitting, etc.

Mangan and Whitaker proposed an approach using the watershed algorithm for partitioning 3D surface meshes [2], and later Razdan and Bae extended the previous approach by proposing a hybrid approach using voxel-based and watershed segmentation methods to segment feature of triangle meshes [3]. In both methods, total curvature is computed and used in segmentation. Initial segments are set up from local minima of the total curvature. However, in some objects, local minima of the total curvature is not easy to find. Therefore, a new way to initialize segments is proposed.

In our method, segments are initialized by applying a mesh growing method to vertices, based on the surface type information. We use Gaussian curvature and mean curvature to classify the surface type of vertices. Gaussian curvature and mean curvature are locally computed by discrete differential-geometry operators. Geodesic distance is computed by our proposed method such that computation of every pair's shortest path of the whole mesh is avoided. In Section 2, we introduce surface mesh analysis for curvatures and surface classification. In Section 3, we explain in details about our segmentation algorithm. In Section 4, we present experiments and show results. Finally in Section 5, we present the conclusions.

2 Surface Analysis

In surface analysis, we compute Gaussian curvature and mean curvature, and later use them to classify the surface type of vertices. The Gaussian curvature K at a vertex point is computed from its adjacent triangles [4, 5].

$$K = \frac{\rho \Delta\theta}{A}, \quad \Delta\theta = 2\pi - \sum_i \theta_i, \quad \text{and} \quad A = \sum_i A_i,$$

where A is the total area of the adjacent triangles T_i, for $i = 1, 2, 3, \ldots$, and ρ is a constant 3. Figure 1 shows example of curvature approximation at vertex P_0.

The mean curvature is defined by the divergence of the surface around the normal vector, $H = \nabla\vec{n}$. The mean curvature normal for a surface mesh is computed as [5, 6, 7],

$$-H\vec{n} = \frac{1}{4A} \sum_{j \in N(i)} (\cot\alpha_j + \cot\beta_j)(P_j - P_i),$$

where $N(i)$ is the vertex P_i's adjacent polygon set, $(P_j - P_i)$ is the edge e_{ij}, α_j and β_j are two angles in $(j+1)^{th}$ and $(j-1)^{th}$ element in $N(i)$ opposite to the edge e_{ij}, respectively, and A is the sum of the areas of triangles in $N(i)$. Figure 2 shows the approximation of mean curvature at vertex P_i.

The surface type of a point can be classified using Gaussian curvature and mean curvature using Besl and Jain

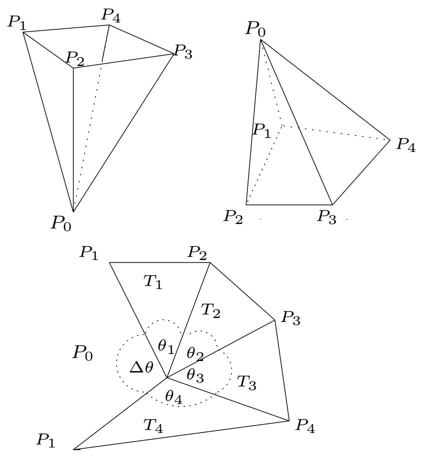


Figure 1. Approximation of Gaussian curvature at vertex P_0

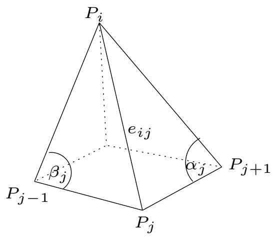


Figure 2. Approximation of mean curvature at vertex P_i

in [8]. The surface type, T, is defined as

$$T = 1 + 3(1 + sgn(H, \epsilon)) + (1 - sgn(K, \epsilon)),$$

where sgn is a tolerance signum function,

$$sgn(x, \epsilon) = \begin{cases} +1 & : \quad x > \epsilon, \\ 0 & : \quad |x| \leq \epsilon, \\ -1 & : \quad x < \epsilon. \end{cases}$$

Table 1 shows the relationship of K, H and T.

	$K > 0$	$K = 0$	$K < 0$
$H < 0$	Peak T=1	Ridge T=2	Saddle Ridge T=3
$H = 0$	none T=4	Flat T=5	Minimal Surface T=6
$H > 0$	Pit T=7	Valley T=8	Saddle Valley T=9

Table 1. Surface types and curvature signs

3 Surface Mesh Segmentation Algorithm

Suppose $\mathbf{M}$ be the mesh structure of objects $\mathbf{O}$, and $\mathbf{M}$ consists of two lists V and E.

$$V = \{v_i | i = 1, ..., N_v\},$$

$$E = \{e_i | i = 1, ..., N_e\}, e_i = \{v_p, v_q\},$$

where v_i are vertices, $1 \leq p, q \leq N_v$ and $p \neq q$. V is a list of all vertices v_i in the mesh. E is a list of edges e_i, which connect two vertices. N_v and N_e are the number of vertices and the number of edges in $\mathbf{M}$, respectively.

In our algorithm, we define four types of segments: (i) peak-type, (ii) pit-type, (iii) minimal surface-type, and (iv) flat-type. The peak-type segment contains vertices with the peak, ridge and saddle ridge surface types. The pit-type segment contains vertices with the pit, valley and saddle valley surface types. The minimal surface-type segment only contains vertices with the minimal surface type. The flat-type also contains vertices with only the flat surface type. Our proposed method consists of three phases: (1) segment initialization, (2) computation of segment centers, (3) assigning segments to vertices and optional segment merging. Details are explained below.

3.1 Segment Initialization

In this step, peak-type, pit-type, minimal surface-type, and flat-type segments are initially formed from consecutive vertices with peak, pit, minimal surface, and flat surface types, respectively. Note that other surface types of vertices are not considered in this step. We apply a region growing based algorithm for this segment initialization. Figure 3 presents the segment initialization algorithm. In the algorithm, there are two functions, **Segment_Initializing** and **Mesh_Growing**. **Segment_Initializing** is first called to set up *vertexSegmentList* array elements to a non-id number and then iteratively call the **Mesh_Growing** function for applicable vertices, which have not been assigned to any initial segments yet. The **Mesh_Growing** function recursively calls itself to find consecutive vertices with the same given surface type. At last a set of initial segments is generated. Each initial segment contains connected vertices with either peak, pit, minimal surface, or flat surface types.

3.2 Segment Center Computation

Suppose S_k be an initial segment containing a set of N_{vk} vertices, V_k. Let $v_{ck} \in V_k$ be the center vertex of S_k. The center vertex is estimated from the vertex that gives the smallest average of its geodesic distances to other vertices $\in V_k$. Since the average and sum are equivalent, v_{ck} is the center vertex such that $\sum_i ||v_{ck}v_{ik}||$ is minimized for all $v_{ik} \in V_k$, and $||.||$ denotes the geodesic distance. The geodesic distance is approximated from the shortest path of vertices. To solve the problem, we first create a weight matrix [9] that not only can represent mesh connectivity of V_k,

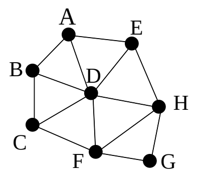
```
Segment_Initializing()
    Let vertexSegmentList be an array of $N_v$ segment ids, and
    sid be a segment id, where $N_v$ is the number of vertices. Let
    UNMARK be a negative constant number for un-segmented
    vertices.

  STEP 1  sid = 1
  STEP 2  for each $v_i \in V$,
              • vertexSegmentList$[i]$ $=$ UNMARK
  STEP 3  for each $v_i \in V$, which has either peak, pit, minimal
          surface, or flat surface types,
              • Let sti be the surface type of $v_i$
              • if (vertexSegmentList$[i]$ $=$ UNMARK)
                Mesh_Growing($v_i$, sid, sti)
              • sid = sid + 1

Mesh_Growing($v_i$, sid, sti)
  STEP 1  vertexSegmentList$[i]$ =sid
  STEP 2  for each $v_n \in V$, which $v_n$ is a neighbor of $v_i$, or
          $\{v_i, v_n\} \in E$,
              • Let stn be the surface type of $v_n$
              • if (stn $=$ sti) and (vertexSegmentList$[n]$ $=$ UNMARK)
                Mesh_Growing($v_n$, sid, sti)
```

Figure 3. Segment Initialization Algorithm

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**
A	0	1	∞	1	1	∞	∞	∞
B	1	0	1	1	∞	∞	∞	∞
C	∞	1	0	1	∞	1	∞	∞
D	1	1	1	0	1	1	∞	1
E	1	∞	∞	1	0	∞	∞	1
F	∞	∞	1	1	∞	0	1	1
G	∞	∞	∞	∞	∞	1	0	1
H	∞	∞	∞	1	1	1	1	0



	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**Σ**
A	0	1	2	1	1	2	3	2	12
B	1	0	1	1	2	2	3	2	12
C	2	1	0	1	2	1	2	2	11
D	1	1	1	0	1	1	2	1	8
E	1	2	2	1	0	2	2	1	11
F	2	2	1	1	2	0	1	1	10
G	3	3	2	2	2	1	0	1	14
H	2	2	2	1	1	1	1	0	10
Σ	12	12	11	8	11	10	14	10	88

Figure 4. Example of the center vertex of a graph (middle) and its weight matrix before (above) and after (below) applying the shortest path algorithm. We assume that the distance between neighboring vertices is equal to one. D is the center vertex which gives the smallest sum of shortest distances to other vertices equal to 8.

but also can represent Euclidean distances between neighboring vertices. The Euclidean distance between two vertices $v_{ik}(x_{ik}, y_{ik}, z_{ik})$ and $v_{jk}(x_{jk}, y_{jk}, z_{jk})$ is computed as

$$|v_{ik}v_{jk}| = \sqrt{(x_{ik} - x_{jk})^2 + (y_{ik} - y_{jk})^2 + (z_{ik} - z_{jk})^2}.$$

Let $\mathbf{A_k}$ be the $N_{vk} \times N_{vk}$ weight matrix of V_k, and $\mathbf{A_k}[v_{ik}, v_{jk}]$ be matrix elements of $\mathbf{A_k}$, where $v_{ik}, v_{jk} \in V_k$. $\mathbf{A_k}$ is thus defined as

$$\mathbf{A_k}[v_{ik}, v_{jk}] = \begin{cases} |v_{ik}v_{jk}|, & \text{if} \quad \{v_{ik}, v_{jk}\} \in E, \\ \quad 0 \quad , & \text{if} \quad v_{ik} = v_{jk}, \\ \quad \infty \quad , & \text{otherwise,} \end{cases}$$

where $|.|$ is the Euclidean distance, and E is the edge list.

We then apply the Dijkstra algorithm [10] to $\mathbf{A_k}$ in order to compute shortest paths between all pairs of vertices in V_k. After applying the shortest path algorithm, $\mathbf{A_k}[v_{ik}, v_{jk}]$ now contain the shortest distance, which is the estimated geodesic distance between vertices v_{ik} and v_{jk}. The center vertex, v_{ck}, is thus the vertex that gives the smallest sum of distances along either row or column of matrices $\mathbf{A_k}$.

$$v_{ck} = \left\{ v_{ik} \;\middle|\; \min \sum_{j=1}^{N_{vk}} \mathbf{A_k}[v_{ik}, v_{jk}] \right\},$$

where $\mathbf{A_k}[v_{ik}, v_{jk}]$ is the geodesic distance between vertices v_{ik} and v_{jk}, and N_{vk} is the number of vertices in V_k. Figure 4 illustrates an example of the center vertex of a graph.

3.3 Segment Assignment

In this step, each unlabeled vertex v_i is assigned to an applicable segment S_k such that the geodesic distance between the segment center v_{ck} and v_i is minimized. Note that unlabeled v_i means *vertexSegmentList*$[i]$ $=$ **UNMARK**. Figure 5 shows steps of assigning v_i to corresponding segments. The first step is to create a list $\mathbf{D_c}$. $\mathbf{D_c}$ contains Euclidean distances from v_i to all v_{ck} whose segment types are valid to v_i as defined at the beginning of the section. In case that there are no initial segments that are valid to v_i, $\mathbf{D_c}$ is empty. Say, in a mesh, there are no vertices with the pit surface type, but there are vertices with valley and saddle valley types. In this case, initially pit-type segments are not created, but new pit-type segments are created later in this phase from vertices with valley and saddle valley surface types.

In the second step, $\mathbf{D_c}$ is checked. If $\mathbf{D_c}$ is empty, then a new segment including v_i is created, and it is added to the initial segment list. However, if $\mathbf{D_c}$ is not empty, we proceed to the next step. In the third step, we select a distance threshold, $dist\_p$, for generating a partial mesh in the next step. $dist\_p$ is selected from the p-th smallest distance of $\mathbf{D_c}$, where at first p is equal to 1. In case that p is increased until it is larger than the number of elements in

$\mathbf{D_c}$, $dist\_p$ is increased by one, and p is decreased by one.

In the fourth step, a new list of vertices, V_t, is created by including all vertices that are within the Euclidean distance $dist\_p$ from the vertex v_i. Then in the fifth step, the partial mesh of V_t is checked for connectivity. First we create the adjacency matrix of V_t to represent connectivity among vertices in V_t. Suppose v_{it} be vertices in V_t, where $i = 1, \ldots, N_t$, and N_t is the number of vertices in V_t. Let $\mathbf{A_t}$ be the $N_t \times N_t$ adjacency matrix of V_t, and elements of $\mathbf{A_t}$ are defined as

$$\mathbf{A_t}[v_{it}, v_{jt}] = \begin{cases} 1, & \text{if} \quad \{v_{it}, v_{jt}\} \in E, \\ 0, & \text{otherwise.} \end{cases}$$

The depth-first search algorithm is applied to $\mathbf{A_t}$ by letting v_i be the root node. Therefore, we get two sets of vertices which are connected and not connected to v_i. Vertices in both the sets can be segment center vertices, labeled vertices, and unlabeled vertices. We then delete all vertices, which are not connected to v_i, from V_t. For connected vertices, we check whether there are segment centers that are connected to v_i; if there are, we proceed to the sixth step, otherwise, we update p by one and go back to the third step. In the sixth step, we apply the Dijkstra algorithm to V_t to compute the shortest path from v_i to other segment centers and other vertices. We then assign v_i to the applicable segment whose segment center in V_t has the smallest geodesic distance to v_i.

After every vertex is assigned to a segment, we check whether all vertices in each segment are connected by applying the depth-first search algorithm again with the segment center as the root node. Non-connected vertices are discarded from their segments. This can happen when vertices do not have any valid initial segments that are close to them. Therefore, we handle them by applying the mesh growing algorithm to group vertices with the similar type together. Recall that in some cases result segments are too small to be used in further analysis. These small segments may be discarded, or they can be merged with one of its neighboring segments such that the difference of the total curvature of two segments is less than a threshold. The total curvature of a segment is the average of the total curvature of all vertices in that segment, and the total curvature of a vertex is computed as in [2].

4 Experiments

The method was tested to segment the surface mesh of 3D structural models of proteins and 3D synthetic objects. The 3D mesh structure of each protein is reconstructed by applying the marching cube algorithm [1] to the 3D volume of the protein structure. The 3D volume is generated from the solvent accessible surface information and atom coordinates of the protein received from the protein data bank [11]. For 3D synthetic objects, we created deformed objects, which contain salient features, e.g., bumps, pits, and saddles, so we can easily see whether our method can seg-

Let v_i be unlabeled vertices in V, and S_k be initial segments from the initial process; $k = 1, \ldots, N_s$, where N_s is the number of initial segments.

STEP 1 compute Euclidean distances between v_i and each v_{ck}, where v_{ck} are centers of segments S_k that have segment types corresponding to v_i; so we get a list of distances

$$\mathbf{D_c} = \{d_{c1}, d_{c2}, ..., d_{cn}\},$$

where n is the number of initial segments.

STEP 2 check whether there are applicable segments for v_i, if there are no applicable initial segments, assign a new segment

- **if** $\mathbf{D_c} \neq \emptyset$, **then** continue to **STEP 3**
- **else** assign v_i to a new segment, add the new segment to the initial segment list (so $N_s = N_s{+}1$) and **return**

STEP 3 let *dist_p* be the p-th smallest distance in $\mathbf{D_c}$ (p is initially set to 1). We can apply an $O(n \log n)$ sorting algorithm to sort $\mathbf{D_c}$, and pick the p-th smallest one for *dist_p*; or we can apply the selection algorithm to get *dist_p*, where the average time complexity is $O(n)$ [10]. In case that $p > n$,

$$\textit{dist\_p} = \textit{dist\_p} + 1,$$

$$p = p - 1.$$

STEP 4 create a new list of vertices, V_t, such that

$$V_t = \{v_j | |v_j v_i| < \textit{dist\_p}\},$$

where $|.|$ is the Euclidean distance, and $v_j \in V$ of the mesh. Note that V_t also includes p segment center vertices whose distances in $\mathbf{D_c}$ are smaller than *dist_p*.

STEP 5 check whether there exist segment centers in V_t which are reachable from v_i by applying the depth-first search algorithm [10] to V_t

- **if** there exist segment centers that are reachable from v_i, **then** discard any vertices in V_t that are unreachable to v_i, and continue to **STEP 6**
- **else** advance p by one so that more vertices can be included in V_t, and goto **STEP 3**

Note that even though by the Euclidean distance segment centers in V_t are close to v_i, they may not be close by geodesic distance, and they may not be connected through the local vertex connectivity in V_t.

STEP 6 apply the Dijkstra algorithm to V_t, while v_i is considered as the source node; then we get the smallest distances from all vertices in V_t to v_i. Therefore, we can now assign v_i to the applicable segment whose segment center has the smallest geodesic distance to v_i.

Figure 5. Segment Assignment Algorithm

	$K > 0$	$K = 0$	$K < 0$
$H < 0$	Peak (Red)	Ridge (Yellow)	Saddle Ridge (Orange)
$H = 0$	none -	Flat (White)	Minimal Surface (Pink)
$H > 0$	Pit (Green)	Valley (Blue)	Saddle Valley (Light Green)

Table 2. Surface types and colors

OBJECTS	# of Triangles	# of Vertices	# of Edges	# of Segments	Time (sec)
1A2Y	1752	878	2628	42	3.796
1EES	3008	1506	4512	74	6.044
OBJ_1	2262	1144	3405	31	5.621
OBJ_2	2212	1119	3330	39	5.513

Table 3. Mesh information and execution times of segmentation

ment these features. We used a 3D graphic modeling software called Maya^{TM} to create all synthetic objects.

Table 3 shows the following information of each object: (i) the number of triangles, (ii) the number of vertices, (iii) the number of edges, (iv) the number of resulting segments, and (v) the execution time, running on Sun Workstation Ultra-5. Figure 6 visually displays segmentation results of proteins, and Figure 7 visually illustrates segmentation results of synthetic objects. For each object, the surface is colored differently by (i) surface normals, (ii) surface types, and (iii) surface segments. We color the surface by surface normals of vertices, so it is easy to see the shape of objects. For surface types, different surface types of vertices are represented by different colors. Table 2 represents surface types and their corresponding colors. For segmentation results, we present different segments by different random colors.

From the results, we can see that segments of salient features can be correctly extracted by our method. These segments are very useful for further analysis. For example, in our protein docking application, we use segments of coarse protein surface meshes to compute possible docking configurations of two proteins such that peak and pit segments are matched together.

5 Conclusion

We propose a surface mesh segmentation method, which utilizes differential geometry and geodesic distance information. In our method, the Dijkstra algorithm is locally applied to approximate geodesic distances between vertices. Moreover, the method also uses the depth-first search and the region growing techniques for partition checking and segment initialization, respectively. For curvatures of the surface mesh, they are computed from discrete differential-geometry operations. Our method has been tested with 3D structures of proteins, and 3D synthetic objects. From the experiments, our method can partition salient features of the objects correctly, and gives satisfactory results. Currently our method is being operated as a part of our protein docking system.

6 Acknowledgment

This publication was made possible by NIH Grant Number RR-15588 from the COBRE Program of the National Center for Research Resources.

References

[1] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4), 1987.

[2] A. P. Mangan and R. T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4), 1999.

[3] A. Razdan and M. Bae. A hybrid approach to feature segmentation of triangle meshes. *Computer-Aided Design*, 2002.

[4] B. Falcidieno and M. Spagnuolo. Polyhedral surface decomposition based on curvature analysis. In *Proc. of the Int. Workshop on Modern Geometric Computing for Visualization*, pages 263–269. Springer-Verlag, 1992.

[5] L. Zhou and A. Pang. Metrics and visualization tools for surface mesh comparison. http://www.cse.ucsc.edu/research/slvg/mesh.html.

[6] M. Desburn, M. Meyer, P. Schröder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH99*, pages 317–324. Addison Wesley, 1999.

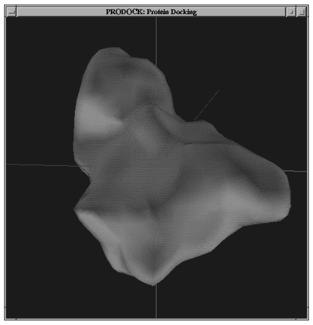
[7] R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design*, 18(4), May 2001.

[8] P. J. Besl and R. C. Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2), 1988.

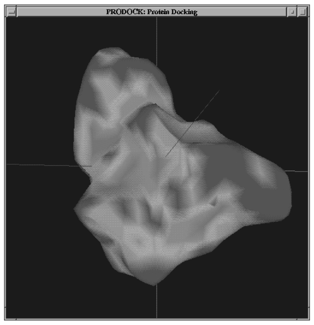
[9] R. Diestel. *Graph Theory*. Springer, 2000.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

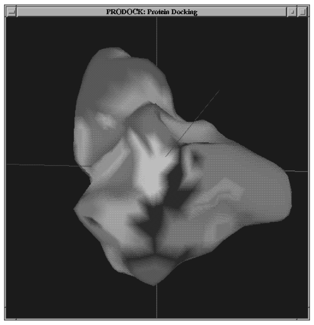
[11] Protein data bank. http://www.rcsb.org/pdb/.



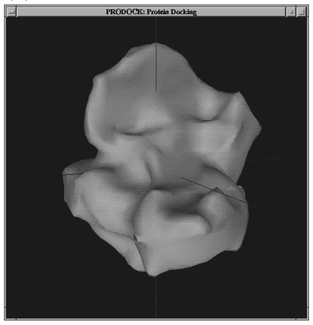
(a). 1A2Y - Surface normals



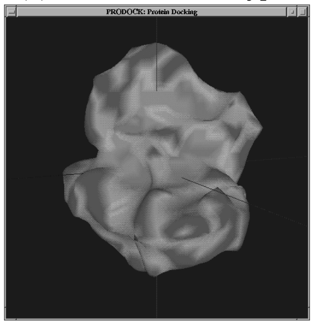
(b). 1A2Y - Surface types



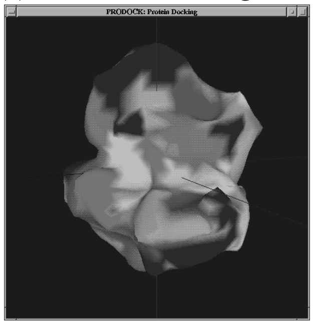
(c). 1A2Y - Surface segments



(d). 1EES - Surface normals

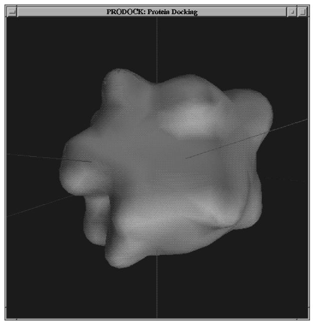


(e). 1EES - Surface types

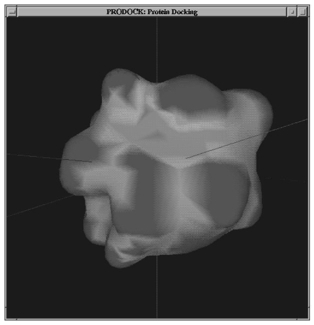


(f). 1EES - Surface segments

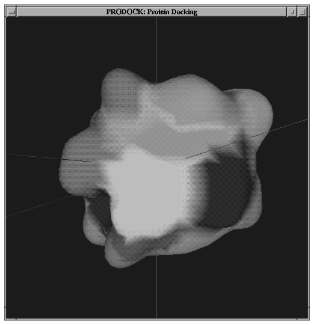
Figure 6. Segmentation results of 3D protein structures



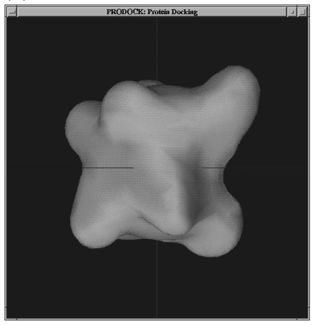
(a). OBJ_1 - Surface normals



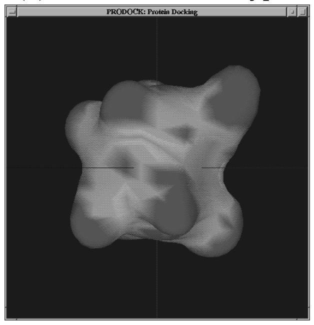
(b). OBJ_1 - Surface types



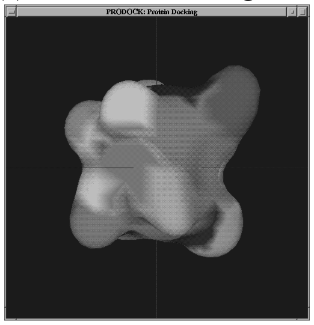
(c). OBJ_1 - Surface segments



(d). OBJ_2 - Surface normals



(e). OBJ_2 - Surface types



(f). OBJ_2 - Surface segments

Figure 7. Segmentation results of synthetic objects