Construction of 3D Orthogonal Cover of a Digital Object

Nilanjana Karmakar[1], Arindam Biswas[1],
Partha Bhowmick[2], and Bhargab B. Bhattacharya[3]

[1] Department of Information Technology
Bengal Engineering and Science University, Shibpur, Howrah, India
nilanjana.nk@gmail.com, abiswas@it.becs.ac.in
[2] Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur, India
bhowmick@gmail.com
[3] Advanced Computing and Microelectronics Unit
Indian Statistical Institute, Kolkata, India
bhargab@isical.ac.in

Abstract. The orthogonal cover of a 3D digital object is a minimum-volume 3D polytope having surfaces parallel to the coordinate planes, and containing the entire object so as to capture its approximate shape information. An efficient algorithm for construction of such an orthogonal cover imposed on a background grid is presented in this paper. A combinatorial technique is used to classify the grid faces constituting the polytope while traversing along the surface of the object in a breadth-first manner. The eligible grid faces are stored in a doubly connected edge list, using which the faces are finally merged to derive the isothetic polygons parallel to the coordinate planes, thereby obtaining the orthogonal cover of the object. The complexity of the cover decreases with increasing grid size. The algorithm requires computations in integer domain only and runs in a time linear in the number of voxels constituting the object surface. Experimental results demonstrate the effectiveness of the algorithm.

Keywords: 3D orthogonal cover, DCEL, isothetic polygon, orthogonal polytope, shape analysis.

1 Introduction

Characterization and feature extraction of 3D objects have been an important aspect of 3D image analysis. Several works have been reported for constructing a polyhedron P enclosing a set of integer points S, usually known as the *discrete volume polyhedrization* [6,7]. In general, these algorithms follow the principle of *marching cube* that generates a triangulated polyhedral surface in which local configurations for voxels are modeled by small triangles. However, such an algorithm bears a limitation that the number of triangular faces in the surface happen to be comparable with the number of points in S. Another work on 3D

J.K. Aggarwal et al. (Eds.): IWCIA 2011, LNCS 6636, pp. 70–83, 2011.

objects relates to digitizing the shape of an object based on range images [21], which involves combining a collection of range images, captured by the range scanner, to form a polygonal mesh that completely describes the object. A number of such meshes aligned and zippered together forms a continuous surface that correctly captures the topology of the object. Based on the surface representation, different techniques for shape analysis of digitized objects form an important area of research in the 3D domain [8,10,13,19,20].

The marching cubes algorithm triangulates a 3D iso-surface based on cubic cell decomposition, their local configurations, and displacement [7,16,17]. The surface intersects a particular cube at certain edges such that the data value of some of the vertices of the cube is greater than or equal to that of the surface. These vertices lie on or inside the surface while other vertices do not. An edge having two opposite type of vertices, thus classified, is intersected by the surface at locations determined by linear interpolation of the vertices. As a cube consists of eight vertices, each in one of two states (inside or outside the surface), a surface can intersect the cube in at most $2^8 = 256$ possible ways, which are reduced to 14 possible patterns using complementary and rotational symmetries. Such an intersection produces at least one and at most four triangles within the cube. Once the intersections are obtained for all cubes, a *marching* procedure through the subsequent cubes leads to an approximate representation of the iso-surface by a triangular mesh. However, since adjacent cubes in the configuration share edges and vertices, the calculation of edge intersection for each cube can be reduced to three edges instead of twelve edges [16]. Approaches to reduce computational activities have been suggested by several researchers [9,15,23]. Nevertheless, the algorithm can be enhanced to handle multi-resolution rectilinear data [22] and data sets in higher dimensional space [2,3].

This paper presents an efficient algorithm that derives the orthogonal cover of a 3D object in $\mathbb{Z}^3$. The problem of construction of inner and outer isothetic/orthogonal covers/frontiers of digital objects in 2D has already been studied in great details [4,5,14]. It is implemented by constructing a doubly connected edge list [1] that maintains complete records of vertices, edges, and faces of the isothetic polygons constituting the cover. Merging the coplanar and contiguous faces of the orthogonal cover begets a compact representation and captures the complete topology without triangulation or any such form of surface decomposition. The algorithm does not consider self-intersecting surfaces or holes lying inside the object, although it can be extended to consider these aspects. Due to the complex topology of a digital object, the 3D orthogonal cover is likely to contain many external concavities, which are gradually revealed as we decrease the grid size, which is a specialty of the proposed algorithm.

The paper is organized as follows. Section 2 contains preliminary definitions and the problem statement. Section 3 presents the proposed algorithm with its step-by-step explanation and time complexity. Experimental results and concluding remarks are given in Section 4.

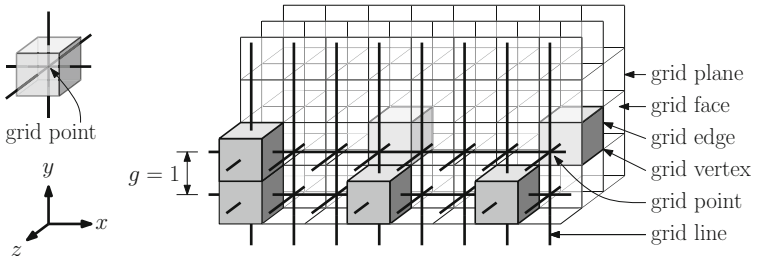

Fig. 1. 3D digital space and 26N [14]. Left: A 3-cell and its corresponding grid point. Right: Three pairs of α-adjacent 3-cells for $\alpha \in \{0, 1, 2\}$, $\alpha \in \{0, 1\}$, and $\alpha = 0$ (from left to right). The 3-cells in each of these three pairs are connected in 26N.

2 Definitions and Preliminaries

Let A be a *3D digital object*, which is defined as a finite subset of $\mathbb{Z}^3$, with all its constituent points (i.e., voxels) having integer coordinates. Each voxel is equivalent to a *3-cell* centered at the concerned integer point. As shown in Fig. 1, two 3-cells can be α-adjacent for $\alpha = 0, 1, 2$. Two 3-cells with centers at $(x_1, y_1, z_1) \in \mathbb{Z}^3$ and $(x_2, y_2, z_2) \in \mathbb{Z}^3$ are said to be connected in *26-neighborhood* (26N) if they are $\alpha(\geq 0)$-adjacent, i.e., $\max(|x_1 - x_2|, |y_1 - y_2|, |z_1 - z_2|) \leq 1$. In our work, the voxels/3-cells constituting the object A are connected in 26N [14].

To derive the orthogonal cover of A, we define $\mathbb{G} := (\mathbb{G}_{yz}, \mathbb{G}_{zx}, \mathbb{G}_{xy})$ as the underlying *grid*. It consists of three orthogonal sets of equi-spaced *grid lines*, namely $\mathbb{G}_{yz}$, $\mathbb{G}_{zx}$, and $\mathbb{G}_{xy}$, their respective grid lines being perpendicular to yz-, zx-, and xy-planes (Fig. 1). The distance between the two consecutive grid lines of $\mathbb{G}_{yz}$ ($\mathbb{G}_{zx}$ or $\mathbb{G}_{xy}$) is defined as the *grid size*, g, which is a positive integer. The point of intersection of three orthogonal grid lines is termed as the *grid point*. Observe that for $g = 1$, the grid $\mathbb{G}$ essentially corresponds to $\mathbb{Z}^3$, as shown in Fig. 1. Further, as each grid point p is equivalent to a 3-cell c_p centered at p for $g = 1$, each face of c_p is a *grid face* lying on a *grid plane*, which is parallel to one of the three coordinates planes. In particular, all the six faces of any 3-cell lie in the grid planes which are parallel to coordinate planes and are unit distance apart. For $g > 1$, we have cubes of length g each. Each such cube is called a *unit grid cube* (UGC) whose vertices are *grid vertices*, edges constituted by *grid edges*, and faces constituted by grid faces. Each face of a UGC lies on a *grid plane*, which is parallel to one of three coordinates planes. Clearly, the distance of each grid plane from its parallel coordinate plane is an integer multiple of g. A smaller (larger) value of g implies a denser (sparser) grid. For notational simplicity, henceforth we use the notation $\mathbb{G}$ to represent both the grid-model and the cell-model, the meaning being evident from the context.

The *orthogonal cover* of the object A is defined as an orthogonal polytope $P(A, \mathbb{G})$ that tightly circumscribes A. An *orthogonal polytope* is a simple polytope (i.e., each vertex is incident on exactly three edges) with all its vertices as grid vertices, all its edges made of grid edges, and all its faces lying on grid

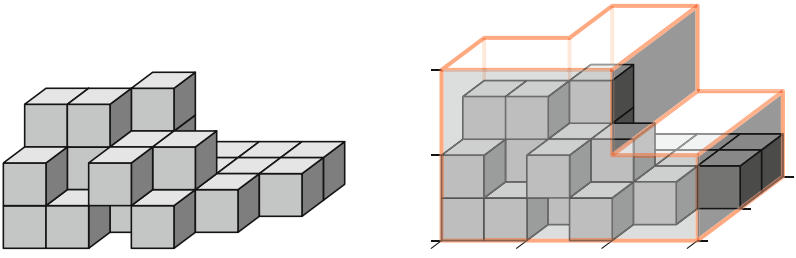

Fig. 2. An object A in $\mathbb{Z}^3$ and its corresponding orthogonal cover for $g = 2$

planes. Each face of an orthogonal polytope is an isothetic polygon whose alternate edges are orthogonal and constituted by grid edges of $\mathbb{G}$.

Problem definition. Given the 3D object A imposed on the grid $\mathbb{G}$, the problem is to construct its orthogonal polytope $P(A, \mathbb{G})$, such that the following conditions are satisfied:

- each point $p \in A$ lies inside $P(A, \mathbb{G})$;
- each vertex of $P(A, \mathbb{G})$ is a grid vertex;
- each edge of $P(A, \mathbb{G})$ is parallel to one of the coordinate axes;
- each face of $P(A, \mathbb{G})$ lies on some grid plane;
- volume of $P(A, \mathbb{G})$ is minimized.

Data structure. A *doubly connected edge list* (DCEL) is a data structure that stores topological information about a 2D subdivision (possibly embedded in 3D space) as a collection of the following records: a) vertex list, b) edge list, and c) face list. The vertex list contains all the vertices of the polytope excluding duplicates. The edge list has all the edges of the polytope in sets of four edges per face of UGC. Each edge is stored as a half-edge (mentioned below) represented by its source vertex and destination vertex. Four consecutive half-edges are assigned the face number representing the face to which the edges belong. For each half-edge $e_{ij} \in f_i$ in the edge list, the plane (yz, zx, or xy) to which the corresponding face f_i is parallel, is also recorded. The edge list also records the pairing of all half-edges (Sec. 3.1). The face list stores the id of a half-edge for each face of the polytope. This half-edge is considered as the first half-edge from which the face can be traversed by referring to the previous and next pointers in the edge list [1,18].

3 Proposed Algorithm

The algorithm OrthoCover3D first computes the start vertex v_s of $P(A, \mathbb{G})$ from the top-left-front point $p_0(i_0, j_0, k_0)$ of A. The object A is stored in a 3D array in which '1's and '0's represent object points and background points, respectively. The grid $\mathbb{G}$ is also represented as a 3D array of structures; each

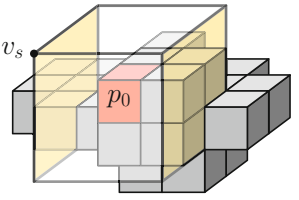

Fig. 3. Defining the start vertex, v_s, from the top-left-front point p_0 (red) of A

structure contains complete information about a UGC—complete and ordered in all respect regarding its eight vertices, twelve edges, and six faces. Observe that p_0 is the top-left-front point of A if and only if for each other digital point $p(i, j, k) \in A$, we have $(k < k_0) \vee ((k = k_0) \wedge (i > i_0)) \vee ((k = 0) \wedge (i = 0) \wedge (j < j_0))$. The point p_0 always lies strictly inside a UGC, whether lying on a grid line, or coinciding with a grid point, or be an ordinary (i.e., non-grid) digital point. From p_0, the coordinates of v_s are obtained as

$$i_s = \lfloor i_0/g \rfloor \times g,\ j_s = \lceil j_0/g \rceil \times g,\ k_s = \lceil k_0/g \rceil \times g. \tag{1}$$

After obtaining v_s, one of the eligible UGC-faces (front face) having v_s as a vertex is enqueued in a queue, Q. Iteratively, each UGC-face f_i is dequeued from Q, and the faces incident on all edges of f_i are checked one by one for their eligibility of belonging to the orthogonal cover, using the EligibleFace procedure. If such a UGC-face is eligible as a part of a polygonal face of the orthogonal cover, then it is enqueued only once in Q. After all the UGC-faces lying close to the object surface are considered, the faces are merged by MergeFace to derive the actual faces of the cover as isothetic polygons parallel to yz-, zx-, and xy-planes. The steps of the algorithm and its related procedures are given below.

Notations: $n_f = \#\text{faces}, n_e = \#\text{edges}, n_v = \#\text{vertices}$; $fid[\cdot], eid[\cdot]$, and $vid[\cdot]$ denote face id, edge id, and vertex id; $start[e]$ denotes the start vertex of an edge e; e_{ij} denotes the jth ($j = 1, 2, 3, 4$) edge of face f_i, and its paired half-edge in f_k (if coplanar with f_i) is denoted by $\bar{e}_{ij}$.

Algorithm. OrthoCover3D$(A, \mathbb{G})$

01. $v_s \leftarrow (i_s = \lfloor i_0/g \rfloor \times g,\ j_s = \lceil j_0/g \rceil \times g,\ k_s = \lceil k_0/g \rceil \times g)$
02. pick the front face f_s of the UGC having v_s as a vertex
03. $n_f \leftarrow\ n_e \leftarrow n_v \leftarrow 0$
04. $fid[f_s] \leftarrow n_f \leftarrow n_f + 1$
05. $color[f_s] \leftarrow$ Gray
06. EnQueue(Q, f_s)
07. **while** Q is not empty
08. $\quad f_i \leftarrow$ DeQueue(Q)
09. $\quad$ **for** each edge $e_{ij} \in f_i$
10. $\quad\quad eid[e_{ij}] \leftarrow n_e \leftarrow n_e + 1$

```
11.          face[e_ij] ← fid[f_i]
12.          if start[e_ij] ∉ V
13.              vid[start[e_ij]] ← n_v ← n_v + 1
14.              V ← V ∪ {start[e_ij]}
15.          assign start[e_ij], next[e_ij], prev[e_ij]   ▷ from array 𝔾
16.          pair[e_ij] ← eid[e_ij]
17.          E ← E ∪ {e_ij}
18.          for each face f_k incident on (start[e_ij], start[next[e_ij]])
19.              if EligibleFace(f_k) = True
20.                  if color[f_k] = White
21.                      fid[f_k] ← n_f ← n_f + 1
22.                      color[f_k] ← Gray
23.                      EnQueue(Q, f_k)
24.                  else if color[f_k] = Black and f_i and f_k are coplanar
25.                      pair[e_ij] ← eid[ē_ij]   ▷ ē_ij ∈ f_k
26.      edge[f_i] ← eid[e_i1]   ▷ e_i1 is the 1st edge of f_i
27.      color[f_i] ← Black
28.      F ← F ∪ {f_i}
29.  MergeFace(F, E)
```

Procedure. EligibleFace(f_k)

```
01. var ← var2 ← False
02. if f_k intersects A
03.     return False
04. else if there exists a point p of A in UGC_1
05.     var1 ← True   ▷ UGC_1 is the left UGC of f_k
06. else if there exists a point p of A in UGC_2
07.     var2 ← True   ▷ UGC_2 is the right UGC of f_k
08. if (var1 = True and var2 = False) or (var1 = False and var2 = True)
09.     return True
```

Procedure. MergeFace(F, E)

```
01. for each face f_i ∈ F
02.     e_ij ← edge[f_i]   ▷ 1st edge of f_i ∈ F
03.     count ← 4   ▷ number of edges to be visited for f_i
04.     do
05.         if eid[e_ij] ≠ pair[e_ij]   ▷ pair[e_ij] exists
06.             ē_ij ← pair[e_ij]
07.             f_k ← face[ē_ij]
08.             next[prev[e_ij]] ← next[ē_ij]
09.             next[prev[ē_ij]] ← next[e_ij]
10.             e_t ← next[prev[e_ij]]
11.             face[ē_ij] ← fid[f_i]
12.             F ← F ∖ {f_k}
```

13. $E \leftarrow E \setminus \{e_{ij}, \bar{e}_{ij}\}$
14. $e_{ij} \leftarrow e_t$
15. $edge[f_i] \leftarrow eid[e_{ij}]$ ▷ 1st edge of the modified face f_i
16. $count \leftarrow count + 2$ ▷ 4 edges added to & 2 edges removed from f_i
17. **else** ▷ $pair[e_{ij}]$ does not exist
18. $e_{ij} \leftarrow next[e_{ij}]$
19. $count \leftarrow count - 1$
20. **while** $count \neq 0$
21. **return** (F, E)

3.1 DCEL Construction

The process of constructing the DCEL starts with initialization of all the UGCs of $\mathbb{G}$ (e.g., setting the color of each grid face of $\mathbb{G}$ as WHITE, marking each grid edge as not in E, and marking each grid vertex as not in V). A UGC face is enqueued in Q only once after setting its color to GRAY and assigning it a unique id (Steps 4-6 and Steps 20-23 of ORTHOCOVER3D). Once a face f_i is dequeued from Q after all the faces incident at the edges of f_i have been considered for eligibility, the face f_i is marked as BLACK. The UGC-faces are thus traversed by BFS, as shown in Steps 7-28. For each edge $e_{ij} \in f_i$, each face f_k, incident on e_{ij}, is checked for its eligibility (Step 19). An eligible face is enqueued only when its color is WHITE. Otherwise, if f_k is BLACK and coplanar with f_i, then f_k can be merged with f_i by deleting their common edge, e_{ij}; hence, $pair[e_{ij}]$ is set to the id of the edge $\bar{e}_{ij} \in f_k$ whose start and end vertices are the respective end and start vertices of e_{ij} (Steps 24-25).

For each face in F, we maintain its fid, the id of its incident edge, and the corresponding coordinate plane to which it is parallel. All these attributes are obtained and updated as shown in different steps of the algorithm ORTHOCOVER3D (Steps 4, 21, and 26). The edge information includes the edge id, the face on which it is incident, its start vertex, ids of its next and previous edges, and its paired half-edge, which are updated in Steps 10, 11, 15, 16, and 25. Vertex information consists of the vertex id, and its coordinates obtaind from $\mathbb{G}$, as shown in Steps 12-14. A vertex is included in the vertex set exactly once (Step 12), which is ensured by setting a flag against each vertex recorded in the concerned structure of its UGC maintained in the 3D array corresponding to $\mathbb{G}$, as explained earlier.

The procedure ELIGIBLEFACE checks whether a grid face f_k is eligible for being a face of the 3D orthogonal cover. First, Step 1 intializes two boolean variables, namely $var1$ and $var2$, to FALSE. Step 2 checks whether the grid face f_k contains an object voxel (3-cell). If so, then f_k is not eligible (Step 3; see also Fig. 4(a)). In Step 4, it is checked whether any point $p \in UGC_1$ is an object voxel. If true, then the flag $var1$ is set to TRUE. Note that UGC_1 denotes the unit grid cube lying in between f_k and f_{k-g}, where f_k and f_{k-g} are two parallel consecutive grid faces separated by a distance of g. Similarly, UGC_2 is the unit grid cube lying in between f_k and f_{k+g}. In Step 6, similar checking is done for the set of points inside UGC_2 to find whether any of them belongs to the object.

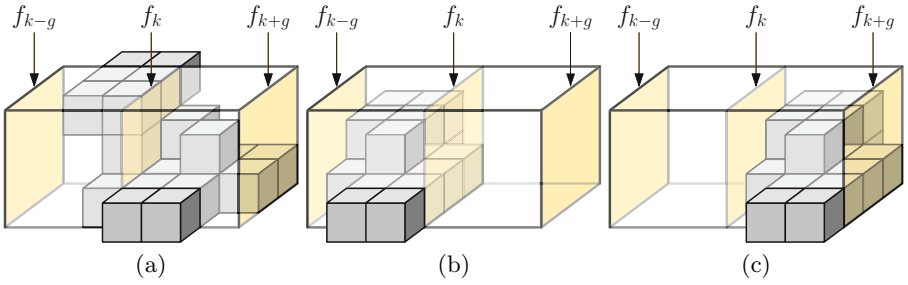

Fig. 4. Testing the eligibility of a face ($g = 3$): (a) f_k intersects the object A; (b) The UGC between f_{k-g} and f_k has object occupancy but no object point is on f_k; (c) The UGC between f_k and f_{k+g} has object occupancy but no object point is on f_k

If no point p in this set belongs to the object then $var2$ remains FALSE. Finally, in Step 8 we ensure that if exactly one adjacent UGC contains object point(s), then only f_k is an eligible face of the orthogonal cover.

3.2 Face Merging

Once the BFS is over, the face list F contains all the grid faces of those UGCs which contain the points on the surface of the object. The MERGEFACE procedure considers each face f_i listed in F (Step 1) and merges all the adjacent faces that are coplanar. We start from the edge e_{ij} listed in F corresponding to f_i, and iteratively follow its next pointer in E to merge the faces. The **do while** loop iterates until the traversal reaches the starting edge corresponding to f_i (Steps 4-21). The variable *count* is used to keep track of the number of edges that are yet to be traversed—as the face merging progresses—to construct a single face as an isothetic polygon of maximal size out of all contiguous UGC-faces that are coplanar with f_i. When $count = 0$ (Step 20), no other face can be merged with the current face f_i, and the next face is considered. If there is a pair of e_{ij}, namely $\bar{e}_{ij}$, such that $pair[e_{ij}] = eid[\bar{e}_{ij}] \neq eid[e_{ij}]$, then it indicates that f_k, the face corresponding to $\bar{e}_{ij}$, is coplanar with f_i (Step 5); hence the face f_k is merged with f_i by deleting e_{ij}, its pair $\bar{e}_{ij}$, and the face f_k (Steps 12-13), and by readjusting the pointers of the related edges (Steps 8-9). Finally, MERGEFACE returns the face list F and the edge list E, where some of the faces have been merged and the first edges of the merged faces have been modified accordingly.

Figure 5 shows a simple example how the grid faces are merged. Let f_1 be selected first from F and e_1 is its start edge, with initialized $count = 4$. As e_1 does not have a paired half-edge whose face is coplanar with f_1, it remains an edge of the face polygon, and the next edge e_2 is considered ($count = 3$). The edge e_2 also does not have a pair ($count = 2$), but its next edge e_3 has a pair, e_5. Hence, e_3 and e_5 are deleted, and the faces f_1 and f_5 are merged, i.e., f_5 is deleted from F, $next[e_2]$ is set to e_6, $next[e_8]$ set to e_4, the face number of all the remaining edges of f_5 are set to $fid[f_1]$, and *count* increases to $2 + 2 = 4$ (Fig. 5(a,b)). In a similar way, as e_6 has e_{12} as its pair, the newly merged face is again merged

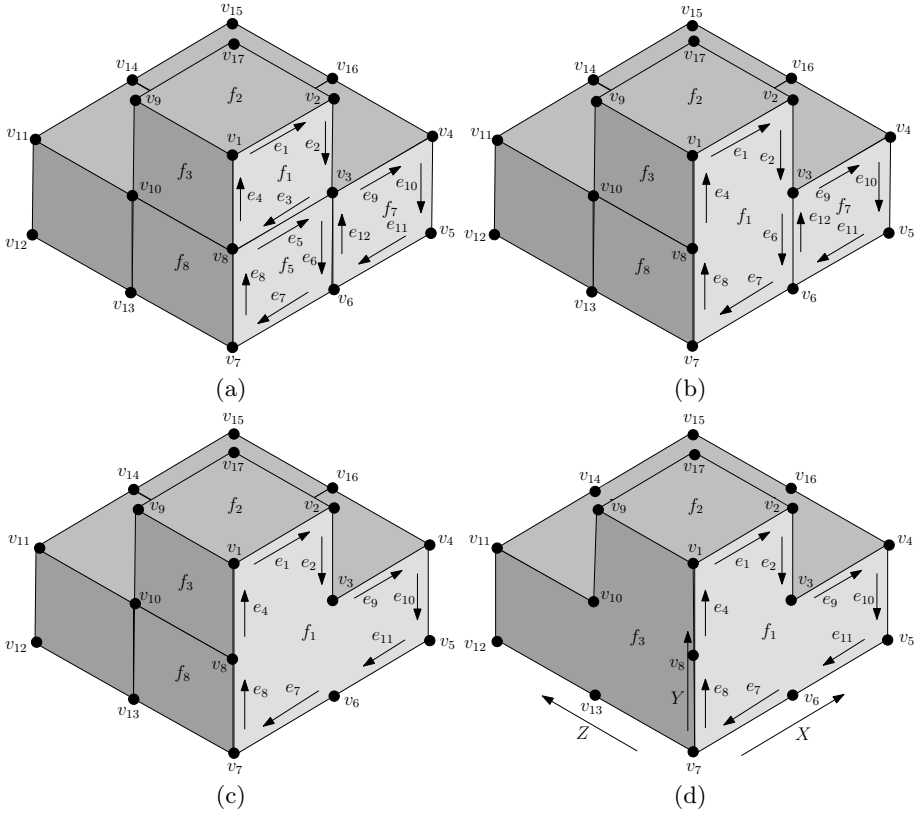

Fig. 5. Demonstration of face merging

with f_7, the face corresponding to the half-edge e_{12}. The edges e_6 and e_{12} are deleted from E, f_7 deleted from F, $next[e_2]$ is set to e_9, $next[e_{11}]$ set to e_7, and *count* increases further to $4 + 2 = 6$ (Fig. 5(b,c)). As the traversal continues, each of e_9, e_{10}, e_{11}, e_7, e_8, and e_4 having no pairing half-edge, are included as edges of the orthogonal cover; *count* falls to $6 - 6 = 0$, and the traversal stops, as e_1 is reached. Thus an orthogonal polygon, $v_1v_2v_3v_4v_5v_6v_7v_8v_1$, is obtained parallel to the xy-plane (Fig. 5(c,d)). Figure 5(d) shows all the merged faces in different planes for the simple orthogonal polytope.

3.3 Time Complexity

We disregard the time required for grid initialization. The **while** loop of the algorithm OrthoCover3D (Steps 7-28) is executed once for each face on the cover. A face f_i is added to Q only if it is an eligible face and not yet been enqueued (Steps 9-11). If enqueued, the face f_i is marked as visited (i.e., colored Gray, Steps 19-23). So, the while loop runs for the number of times equal to the total number of grid faces constituting the orthogonal cover. Let n be

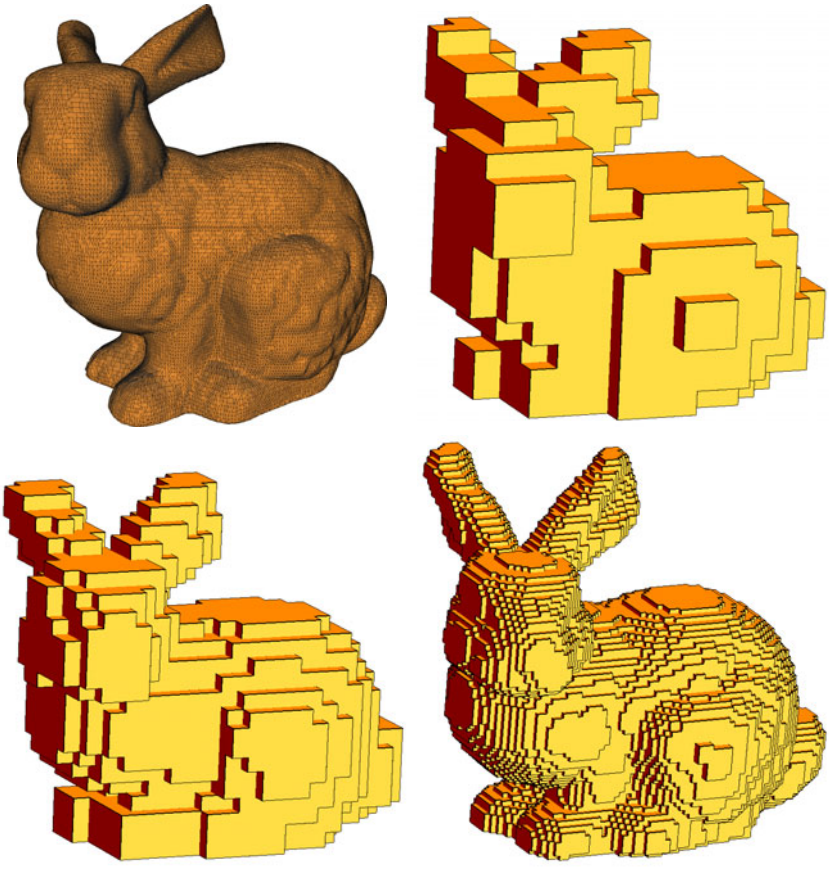

Fig. 6. "Stanford Bunny" and its covers for different grid sizes. Top-left: original object, $n = 58605$. Top-right: $g = 12, n_v = 769, n_e = 1492, n_f = 367$. Bottom-left: $g = 7, n_v = 2040, n_e = 3864, n_f = 903$. Bottom-right: $g = 2, n_v = 22305, n_e = 42896, n_f = 10377$.

the number of voxels constituting the object surface connected in 26N. Then the runtime complexities for the best and the worst cases can be analyzed as follows.

Best Case: Minimum number of UGCs, each UGC being a cube of length g and containing points of the object surface, is $O(n/g^3)$. As each UGC has six faces—a maximum of five of which can be a part of the cover, the number of grid faces on the surface of the object is again $O(n/g^3)$. In each iteration, the eligibility of a grid face f_k for being a face on the orthogonal cover is decided by checking the voxels inside the two UGCs on both sides of (i.e., adjacent to) f_k, which requires $O(1)$ comparisons in the best case (Step 19). So, the **while** loop (Steps 7-28) over all the grid faces comprising the orthogonal cover requires at most $O(n/g^3) \times O(1) = O(n/g^3)$ computations. To check the color of a face, or to check whether a vertex is already included in V, or to check an edge is

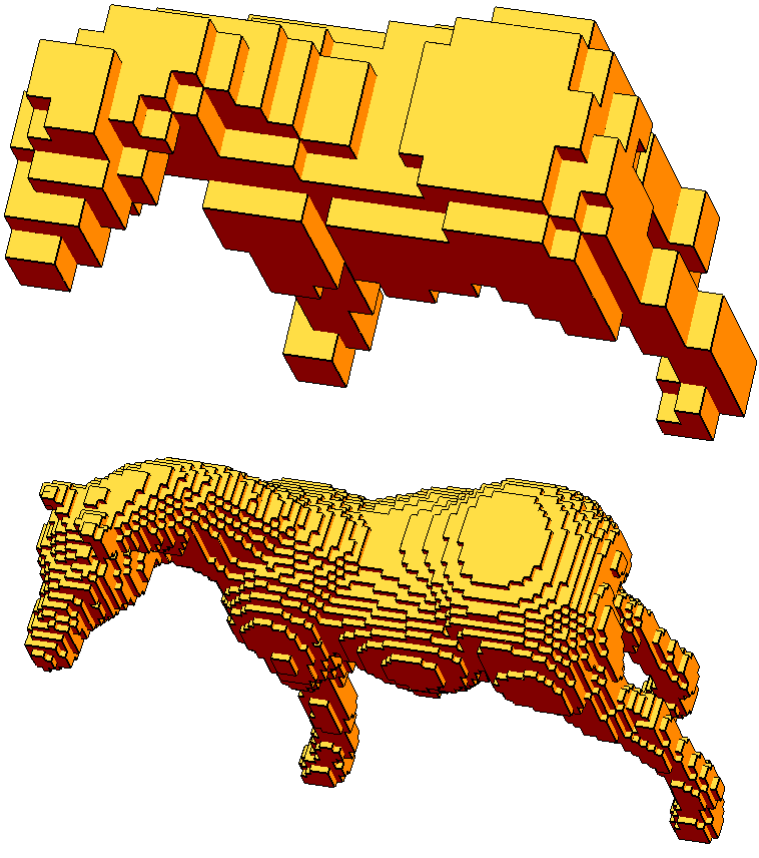

Fig. 7. Results on "Horse". Top: $g = 8, n_v = 1205, n_e = 2328, n_f = 523$. Bottom: $g = 2, n_v = 25512, n_e = 53188, n_f = 13238$.

already in E, we use the grid information as explained earlier. Hence, each of these checks (Steps 12, 20, 24) needs constant time.

Before applying the MergeFace procedure, the total number of edges (half-edges) in E is four times the total number of eligible faces, i.e., $4 \times O(n/g^3) = O(n/g^3)$; the total number of vertices is also $O(n/g^3)$, as each vertex is incident on at most six edges. During merging, the **for** loop (Steps 1-20) considers each face $f_i \in F$ one by one (**do-while** loop: Steps 4-20), and checks the pair of e_{ij}. If $pair[e_{ij}] \neq eid[e_{ij}]$, then it indicates that the face f_k on which $pair[e_{ij}]$ is incident, is coplanar with f_i. So the face f_k is merged with f_i and necessary updates/modifications of the relevant edges and faces are done in constant time. Since the total complexity (number of faces, edges, and vertices) of the DCEL is bounded by $O(n/g^3)$, the time complexity of MergeFace becomes $O(n/g^3)$. So, the best-case time complexity of the algorithm OrthoCover3D is given by $O(n/g^3)$.

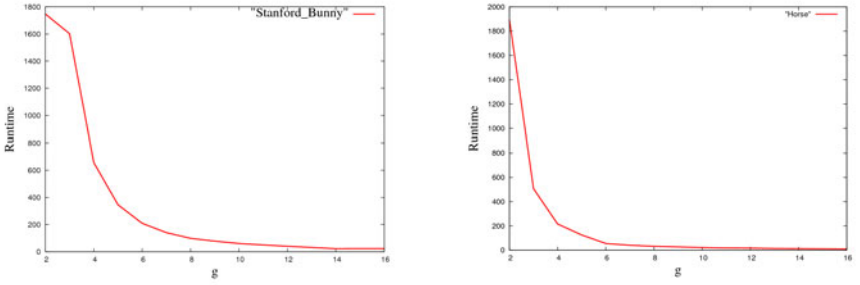

Fig. 8. Runtime (in secs.) versus grid size g. Left: Stanford Bunny. Right: Horse

Worst Case: Each UGC may contain as low as $O(g)$ surface voxels, and so each grid face may require $O(g^3) - O(g) = O(g^3)$ time for checking its eligibility. Maximum number of UGCs can be $O(n/g)$, wherefore we have $O(n/g) \times O(g^3) = O(ng^2)$ time complexity in the worst case for finding the faces of the orthogonal cover. Maximum number of faces would be $O(n/g)$, since each UGC (containing surface voxels) will contribute at least one face and at most five faces to the orthogonal cover. Hence, with a justification similar to the best-case analysis, the time complexity of MergeFace is $O(n/g)$. The overall worst-case time complexity is, therefore, given by $O(ng^2) + O(n/g) = O(ng^2)$.

In practice, however, we find that the runtime decreases rapidly with increasing grid size. This is revealed by our exhaustive experimentation, some of which are presented in Sec. 4. Thus, actual runtime for real-world digital objects in $\mathbb{Z}^3$ tends towards the best case, which is a characteristic of our algorithm.

4 Results and Conclusion

We have implemented the algorithm in C in Linux Fedora Release 7, Kernel version 2.6.21.1.3194.fc7, Dual Intel Xeon Processor 2.8 GHz, 800 MHz FSB. OpenGL running with MinGW on Windows XP Professional is used for the purpose of 3D rendering [11,12]. The algorithm has been tested on several 3D objects for different grid sizes, a few of them being presented in Figs. 6 and 7.

Notice that as g is decreased in Fig. 6, a tighter description of the bunny is obtained, as the numbers of vertices, edges, and faces increase rapidly. Figure 8 shows the runtimes of the algorithm on different objects ("Stanford Bunny" and "Horse") for different grid sizes. For $g = 2$, 8, and 16, the respective CPU times required to construct the orthogonal covers for "Stanford Bunny" are 1747, 98, and 22 seconds, whereas those for "Horse" are 1893, 32, and 10 seconds. Clearly, the runtime falls significantly with the increase in grid size. The description of the object in the form of the orthogonal cover at a lower grid size contains more information (than the one at a higher grid size) about the nature of the object at the cost of more CPU time and storage space.

References

1. Berg, M.D., Cheong, O., Kreveld, M.V., Overmars, M.: Computational Geometry—Algorithms and Applications. Springer, Heidelberg (1997)
2. Bhaniramka, P., Wenger, R., Crawfis, R.: Isosurface construction in any dimension using convex hulls. IEEE Trans. on Visualization and Computer Graphics 10, 130–141 (2004)
3. Bhaniramka, P., Wenger, R., Crawfis, R.: Isosurfacing in higher dimensions. In: Proceedings of Visualization, Salt Lake City, pp. 267–273 (2000)
4. Biswas, A., Bhowmick, P., Bhattacharya, B.B.: Construction of Isothetic Covers of a Digital Object: A Combinatorial Approach. Journal of Visual Communication and Image Representation 21, 295–310 (2010)
5. Biswas, A., Bhowmick, P., Bhattacharya, B.B.: TIPS: On Finding a Tight Isothetic Polygonal Shape Covering a 2D Object. In: Kalviainen, H., Parkkinen, J., Kaarna, A. (eds.) SCIA 2005. LNCS, vol. 3540, pp. 930–939. Springer, Heidelberg (2005)
6. Brimkov, V.: Discrete volume polyhedrization: Complexity and bounds on performance. In: Proc. of the International Symposium on Computational Methodology of Objects Represented in Images: Fundamentals, Methods and Applications, CompIMAGE 2006, pp. 117–122. Taylor and Francis, Coimbra (2006)
7. Coeurjolly, D., Sivignon, I.: Reversible discrete volume polyhedrization using Marching Cubes simplification. In: SPIE Vision Geometry XII, vol. 5300, pp. 1–11 (2004)
8. Cohen-Or, D., Shamir, A., Shapira, L.: Consistent Mesh Partitioning and Skeletonization using the Shape Diameter Function. The Visual Computer 24, 249–259 (2008)
9. Giles, M., Haimes, R.: Advanced interactive visualization for CFD. Computing Systems in Engineering 1, 51–62 (1990)
10. Golovinskiy, A., Funkhouser, T.: Randomized cuts for 3D mesh analysis. ACM Transactions on Graphics (Proc. SIGGRAPH ASIA) 27, Article 145 (2008)
11. Hearn, D., Baker, M.P.: Computer Graphics with OpenGL. Pearson Education Inc., London (2004)
12. Hill, F.S., Kelley, S.M.: Computer Graphics Using OpenGL. Pearson Edcation Inc., London (2007)
13. Katz, S., Leifman, G., Tal, A.: Mesh segmentation using feature point and core extraction. The Visual Computer 21, 649–658 (2005)
14. Klette, R., Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Picture Analysis. Morgan Kaufmann, San Francisco (2004)
15. Livnat, Y., Shen, H.-W., Johnson, C.: A near optimal isosurface extraction algorithm using span space. IEEE Trans. Visualization and Computer Graphics 2, 73–84 (1996)
16. Lorensen, W.E., Cline, H.E.: Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics 21, 163–169 (1987)
17. Newman, T.S., Yi, H.: A Survey of the Marching Cubes Algorithm. Computers & Graphics 30, 854–879 (2006)
18. Preparata, F.P., Shamos, M.I.: Computational Geometry—An Introduction. Spinger, New York (1985)
19. Shapira, L., Shalom, S., Shamir, A., Cohen-Or, D., Zhang, H.: Contextual Part Analogies in 3D Objects. International Journal of Computer Vision 89, 309–326 (2010)

20. Shlafman, S., Tal, A., Katz, S.: Metamorphosis of Polyhedral Surfaces using Decomposition. In: Eurographics 2002, pp. 219–228 (2002)
21. Turk, G., Levoy, M.: Zippered polygon meshes from range images. In: SIGGRAPH 1994, pp. 311–318 (1994)
22. Weber, G., Kreylos, O., Ligocki, T., Shalf, J., Hagen, H., Hamann, B.: Extraction of crack-free isosurfaces from adaptive mesh refinement data. In: Proceedings of VisSym 2001, Ascona, Switzerland, pp. 25–34 (2001)
23. Wilhelms, J., van Gelder, A.: Topological considerations in isosurface generation extended abstract. Computers Graphics 24, 79–86 (1990)