Growing-cube Isosurface Extraction Algorithm
For Medical Volume Data

Tong-Yee Lee, Chao-Hung Lin

Visual System Laboratory (VSL)
Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, TAIWAN, R.O.C

ABSTRACT

In medical applications, three-dimensional volume data such as CT and MRI are gathered from medical-imaging devices. Marching cube (MC) algorithm is a common routine to extract isosurfaces from volume data. The MC algorithm generates the massive number of triangles to represent an isosurface. It is difficult to render this amount of triangles in real-time on general workstations. In this paper, we present a growing-cube algorithm to reduce the number of triangles generated by the MC algorithm. Growing-cube algorithm uses a surface tracker to avoid exhaustive searching isosurfaces cell by cell and therefore it saves computation time. During surface tracking, the growing-cube algorithm adaptively merges surfaces contained in the tracked cells to reduce the number of triangles. Surfaces are merged as long as the error is within user-specified error thresholds. Therefore, the proposed algorithm can generate a variable resolution of isosurfaces according to these error parameters.

Keywords: Marching-cube, Growing-cube, Surface tracking and merging, Meta-surface, Surface triangulation

1. Introduction

In medical applications, major sources of three-dimensional volume data are gathered from medical-imaging devices such as CT, MRI and PET scanners. Generating polygons to approximate surfaces of organs from 3D volume data has been widely used by clinicians to visualize, manipulate and measure three-dimensional internal structures of patients. Many algorithms [1,2,3,4] have been proposed to generate isosurfaces from volume data. Among them, the marching cube (MC) algorithm [4]
has been used as a standard isosurface generation algorithm. Using the MC method, the number of triangles generated per cell varies from one to five. Therefore, for example, for given a 512x512x100 medical CT data, the MC algorithm potentially produces the number of triangles varying from 500K to 2000K. This sheer amount of triangles is still prohibitive to achieving a real-time rendering on most workstations.

In the past, much effort has been made to reduce this amount. The solutions proposed can be roughly classified into two categories. The first class produces triangles using the MC algorithm, and then employs polygon simplification methods to reduce the number of triangles [2,5,6]. In contrast, the second class exploits adaptive techniques [1,7,8,9,10] to reshape the cube size or to down-sample the volume data set where the isosurface is mostly flat. In this paper, the proposed method, called Growing-cube method does not reshape nor down-samples volume cells, but it adaptively generates varying-size surfaces.

Other issues such as ambiguity problem in the method of approximating the surface [11,12,13,14], the redundant polygon generation problem [15], and computational speed improvement [8,10,16] have attracted many researchers to work on them. In this paper, the proposed method can save the number of triangles as well as save computational timings. In some situation, not all components of the isosurface are connected. Bajaj et al. [20] and Westermann et al. [21] resolved this contour following problem. Growing-cube method fails if the isosurface has more than a connected component, since traversal starts from a single seed. Actually, experience with visualizing our medical applications has shown that most datasets have only a connected component. If required, to resolve this situation, multiple-seeds are allowed to interactively insert to extract all components of the isosurface.

The remainder of this paper is organized as follow. In the next section, we will review the most related prior work. Section 3 will introduce each step of the proposed method in details. The experimental results for artificial data sets and medical volume data are described and discussed in Section 4. Finally, we give the summary of this paper in Section 5.

2. Previous Work

Van Gelder et al. [16] employed an Octree data structure to reduce the number of cubes traversed, and thus they improved the MC’s computational efficiency. However,
this approach does not reduce the number of triangles generated by the MC algorithm. In contrast, Ning et al. attempted to use an Octree to assist in decimating the number of triangles [8]. However, this algorithm is not very efficient in computation. Shu et al. [7] proposed an adaptive marching cube (AMC) algorithm to reduce the number of triangles representing the surface. This method is limited to data sets of resolutions of the power of two. The initial cubes are recursively subdivided into eight sub-cubes, if the approximating surface has high curvature. This approach has potential to miss small objects inside the cube. Independently, Muller et al. proposed a similar approach termed as splitting-box algorithm [1]. Li et al. [15] improved MC algorithm in two ways. First, this method simply uses the middle point as the triangle vertices between two adjacent sample points. Second, it detects the redundant triangles and eliminates them. Similarly, Montani et al. [17] proposed a discretized MC method where the edge intersections are approximated by edge midpoints. Furthermore, the discretized MC also builds superfaces and simplifies them. Yagel et al. [9] proposed an Octree-based strategy to decimate the number of triangles of MC. Like [7], this method requires additional stage to deal with crack patching problem. However, this method is better than [7], since its crack patching does not introduce new triangles. Park et al. [10] presented a vertex-merging algorithm that merges the vertices of triangles generated by the MC algorithm at a surface generation step. It is a bottom-to-up approach, and hence, it avoids cracks and missing small objects in cube. In this approach, it still exhaustively searches all cells.

Figure 1. Three components in the proposed Growing-cube algorithm.

3. Growing-cube Algorithm

The growing-cube algorithm consists of three components as shown in Figure 1. The surface tracking searches each cell through which the isosurface passes instead of exhaustive searching in the MC algorithm. The connected surface contained in a searched cell can be merged (or grown) into a large connected surface called meta-surface. The surface tracking and meta-surface merging are executed at the same time. A meta-surface can grow into an arbitrarily large surface as long as the growing condition is met. Once a meta-surface cannot grow, we triangulate it and then start another new meta-surface growing if necessary. Each component of algorithm is explained in detail below.
3.1 Surface Tracking

The MC algorithm generates triangles cell by cell to approximate isosurfaces. The values of grid points and linear interpretation are used to determine where the isosurface intersects an edge of a voxel. In total, there are only 15 topologically different configurations, as shown in Figure 2. Like [9], we employ a surface tracker to identify each cell through which the isosurface passes. The user specifies a seed cell that is not an ambiguous configuration of the MC algorithm and then the surface tracker starts searching seed’s neighboring cells. For each cell, at most, there are six tracking directions. Among 15 cases, those cases 0, 1, 2, 3, 5 and 8 do not need to track six neighboring cells. Yagel et al. [9] pointed that these six cases account for 90% of the cases encountered in extracting an isosurface in their applications. Therefore, the tracking approach leads to more saving in computation. To speed up surface tracking, a look-up table is created and is used to guide which cells to visit from a given cell. To create this table, we label each edge $e_i$ (as shown in Figure 3 (a)) and six tracking directions (as shown in Figure 3 (b)) for a given cell. This look-up table is shown in Figure 4. For example, if a cell is identified as the case 1 of the MC configurations, this cell has three intersected edges, $(e_1,e_2)$, $(e_1,e_{10})$ and $(e_2,e_{10})$. Then, we use these three edge pairs as indexes to retrieve tracking directions from the look-up table. Like $(e_1,e_2)$, table$[e_1,e_2] = 1$ says the current cell needs to track that cell in the direction of ‘1’.
Figure 2. Topologically different configurations of the MC algorithm.

Figure 3. Edge and direction labeling.
3-2 Meta-Surface Growing

The user specifies a seed cell that is not an ambiguous case of MC method and contains connected triangular surfaces like cases 2, 5, 8, 9, 11 and 14. Then, the seed meta-surface attempts to grow into an arbitrary large one by merging neighboring meta-surfaces contained in neighboring cells. Recursively, this meta-surface growth toward tracking directions continues until that the growing criteria are violated. Then, this violated meta-surface can be a new seed to start another meta-surface growing. During growing, the normal of the meta-surface is always used and is computed by finding the normal of the best-fit plane for all triangular patches contained in the cell. Figure 5 assumes that a growing meta-surface $GM$ whose normal is $(N_x, N_y, N_z)$, attempts to merge another meta-surface $F$ whose normal is $(F_x, F_y, F_z)$ contained in a tracked cell. The following criteria must be satisfied for above meta-surface growth.

Figure 4. Look-up table used for surface tracking. Non-existing direction is marked by "*".
Once merge is successful, we compute a new \((N_x,N_y,N_z)\) to represent this new and larger meta-surface.

1. The angle between \((N_x,N_y,N_z)\) and \((F_x, F_y, F_z)\) must be less than a threshold, \(\theta_N\).
2. The shortest distance from each triangular patch of \(F\) to \(GM\) must be less than a selected threshold, \(dist\). The above two conditions guarantee two faces are flat enough.
3. We also require the normal of each vertex contained in \(F\) must be similar to \((N_x,N_y,N_z)\). In other words, the angle between normals must be less than a threshold denoted as \(\theta_V\). The shading effect is related to vertex normals. This condition guarantees there is no too much difference in shading effect between \(F\) and \(GM\). In the proposed approach, we use a meta-surface \(F\) to approximate the connected triangular patches. In some cases such as the case 5, these connected triangular patches are not very flat. It is not correct to use a meta-surface \(F\) to represent them. This condition avoids this error happening.
4. A meta-surface \(F\) (i.e., case 1, 2, 5, 8, 9, 11 or 14) must be connected and then be considered to merge with \(GM\). Disconnected cases are isolated from the meta-face growing and their triangular patches are output directly.
5. Both meta-faces \(F\) and \(GM\) cannot fold over on their approximating plane. Later, we perform surface triangulation in two-dimension rather than three-dimension. We need to compute orthogonal projection of vertices of \(F\) on the plane containing \(GM\). If there is some fold-over occurring, it fails to triangulate the merged surface. This condition is used to guarantee the correctness of meta-surface triangulation in the proposed algorithm.

Merging can start if all of the above five conditions are met. When two faces are allowed to merge, the merging is simply done by eliminating the shared edge as shown in Figure 6. The meta-surface maintains the boundary edges and internal edges are removed. Later, the surface triangulation is employed to adaptively add internal edges into the meta-face. There is an implementation issue about finding the approximated plane. One possible solution is to find the best approximating plane to contain both meta-surfaces \(GM\) and \(F\) for each meta-surface merging. Once a new face is merged, and then a new approximating plane is computed. Generally, this computational time grows linearly with the number of faces being merged. Each surface included in the \(GM\) and \(F\) must be checked against the non-fold-over rule (condition 5). Assume that there are \(n\) faces merged in final, the merging is computed in \(O(n^2)\) time. Therefore, this solution seems not very efficient in computation. To

---

1 Later in this section, we actually use the normal of a seed-meta-surface instead.
reduce this cost, we alternatively use the plane containing the seed face rather than compute a newly approximating plane every-time. From our viewpoint, if the faces can merge with a given seed face, these merged faces should be almost on the same plane containing the seed face. Otherwise, they should not merge. Therefore, we use this solution instead. Now, we do not need to compute an approximating plane whenever a new face is merged. Furthermore, we do not need to re-check all merged faces against the no-fold-over rule every time. In this simplified implementation, only the newly meta-surface $F$ is required to be verified. Therefore, in total, this cost can be reduced from $O(n^2)$ to $O(n)$.

Now, let us proceed to present the growing cube algorithm and its pseudo-code is outlined in Figure 7. In this implementation, we maintain two queues, namely $DirQueue$ and $SeedQueue$. For the $DirQueue$, it records the possible tracking...
directions for the current meta-surface. In case that the current cell cannot merge, this cell is inserted in the SeedQueue instead. Later, once this cell is pop up from the SeedQueue, and then it potentially starts a new meta-surface growing. In this pseudo code, a 3D flag array (i.e., is initialized flag = 0) exactly the size of the 3D dataset is used. The flag value of the cell indicates this cell whether it has been tracked (flag = 1), or the cell was in the DirQueue (flag=2) or that was in the SeedQueue (flag = 3). Therefore, this 3D flag array avoids cells being tracked and queued more than once.

In the inner loop, if the current F satisfies the growing criteria, we perform face merging. Otherwise, we insert this cell into the SeedQueue, and later this cell can potentially start a new meta-surface growing. Whenever the current meta-surface accomplishes its growth (i.e., leave the inner loop), we start triangulating the current meta-surface and output its triangular surfaces. Then, we pop up a new cell from the SeedQueue and enter the inner loop again. The above process will not finish until both the DirQueue and the SeedQueue are empty. Note that once ambiguous cases take place and we employ [22] to solve this ambiguity.

**Algorithm Growing-cube**

Input: a seed cell (i,j,k) that contains an unambiguous configuration of MC algorithm and its triangular patches are connected;

Output: triangular surfaces

Begin

For all tracking directions of a seed cell (i,j,k) using the proposed surface-tracker

1. Insert cells located on tracking directions into the DirQueue;
2. While (SeedQueue != NULL) {
3. While (DirQueue!= NULL) {
4. Remove a new cell termed as (i,j,k), from the DirQueue;
5. Find its MC configuration and determine if it is qualified as a meta-surface F or not (i.e., must be connected)
6. If cell (i,j,k) is an ambiguous case, we solve its ambiguity by the method in [22] and directly output its triangular patches.
7. Then, we insert cells located on tracking direction of cell (i,j,k) into SeedQueue and set flag of cell (i,j,k) to be 1;
8. Else If F is a disconnected case of the MC algorithm, we directly output its triangular surfaces and insert each tracking direction into SeedQueue and set flag of (i,j,k) to be 1;
9. Else If a qualified F exists and all growing criteria are met,

1. The meta-surface F merges with the current
2. Set the flag of \((i,j,k)\) to be 1;
3. For each tracking direction of cell \((i,j,k)\), a neighboring cell say \((i',j',k)\)
   \[\text{If cell } (i',j',k) \text{ has not been tracked (i.e., } flag = 0)\]
   a. insert \((i',j',k)\) in the \textit{DirQueue}
   b. set flag of \((i',j',k)\) to be 2;
   \[\text{// it avoids } (i',j',k) \text{ being tracked more than once; }\]
\]
\] Else If the cell \((i,j,k)\) (i.e., \(flag \neq 3\)) is not in the \textit{SeedQueue}:
\[\text{// * cell } (i,j,k) \text{ does not meet the growing criteria} \]
\[\{\]
   insert cell \((i,j,k)\) in the \textit{SeedQueue};
   set flag of the cell \((i,j,k)\) to be 3
   \[\text{// ** to avoid being inserted more than once**/} \]
\[\}
\]
Triangulate the current \textit{Meta-surface}
Output triangular surfaces;

Figure 7. Growing-cube algorithm.

Figure 8 is an example to show that the proposed algorithm potentially performs better than an Octree-based approach [9]. In this figure, each meta-surface is labeled by a distinct alphabet. The meta-surface \(C\) consists of several small meta-surfaces contained in nine cells numbered by \(C_0, C_1, \ldots\) and \(C_8\), respectively. These nine cells can merge by means of the proposed method. In contrast, the Octree-based approach generates four smaller meta-surfaces, \((C_0, C_1)\), \(C_2\), \((C_3, C_4, C_5, C_6, C_7)\) and \(C_8\), respectively. In this example, there are several similar cases such as meta-surfaces \(M\) and \(N\). Our comment is that the Octree is a good data structure to organize merging process [9], but also limits the freedom of merging. From this respect, the proposed algorithm has an advantage over [9].

3.3 Meta-surface Triangulation
We triangulate meta-surfaces in 2D rather than in 3D and describe this process as follows. First, we project all boundary vertices of a meta-surface on the approximating plane. Here, the plane of a seed meta-surface is used as the approximating plane. Next, we add internal vertices into the meta-surface to form several triangular meshes. To find a suitable triangulation method is essential at this stage. In particular, this triangulation must handle hole problem. For example, in Figure 9, there are several meta-surfaces denoted as $S_1$, $S_2$, ..., and $S_5$ and these meta-surfaces are surrounded by a single neighboring meta-surface $S_0$. In this situation, while triangulating this meta-face $S_0$, we create a hole. To solve this problem, we employ [18] to triangulate the meta-surface. This method was extended [18] from Seidel’s fast polygon triangulation method [19] to handle hole problem.

To directly triangulate the meta-face might be not the best implementation. Figure 10 (a) shows an example of a typical meta-surface after merging. In this example, there are many gear-tooth like edges appearing in the boundary of the meta-surface. These gear-tooth like edges lead to more triangular surfaces. To avoid this, we use a simple poly-line approximation to best fit these edges and thus to reduce the number of gear-teeth as shown in Figure 10 (b). For the poly-line approximation, we need to set up a distance error, $\varepsilon$. If the distance between a gear-tooth and an approximating poly-line is within the distance error, this gear-tooth can be eliminated. After this simplification process, we proceed to triangulate the meta-surface. Considering all cases of MC algorithm, those cases 3, 4, 6, 7, 10, 12 and 13 are with disjointed triangular surfaces. Since these triangular surfaces are located at a given cell, the distance between two surfaces is very small. If this distance is smaller than the threshold $\varepsilon$, the proposed algorithm potentially merges two disjointed surfaces by mistake during this border simplification. For example, in Figure 8, in cell $T_0$, there are two disjointed iso-surface contours. If these two surfaces are merged incorrectly, it becomes a connected isosurface. This is another reason why we require that a given cell can be merged if it is a connected surface configuration of the MC algorithm.
Figure 8. An example of isosurface extraction.

Figure 9. Hole occurs in surface triangulation.
4. Results and Discussions

To evaluate the proposed algorithm, we also implemented the original MC algorithm and executed both algorithms on a Pentium II-300 personal computer\(^2\). The following four datasets were used in our evaluation: (1) 256x256x30 Colon CT dataset (Figure 11), (2) 128x128x56 CT Head dataset (Figure 12), (3) 128x128x128 artificial Stomach dataset (Figure 13) and 16x16x16 volumetric Cube dataset (Figure 14). Table 1 compares the execution time of the surface tracker and the original MC algorithm. As shown in this table, the surface tracker always performs faster than the MC algorithm, and thus the proposed method gains computational saving. Experimental results show that the computational saving might not exactly increase with increasing resolution of datasets. In fact, it depends on the ratio of the number of non-empty cells over the size of volume data. To verify this observation, we also show the ratio information in Table 1. Among these four datasets, the Colon dataset performs the best due to the lowest ratio of non-empty cells. The isosurfaces extracted by both MC and surface tracker are identical for each dataset.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Execution Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Colon (256x256x30)</td>
</tr>
<tr>
<td>MC</td>
<td>3.55</td>
</tr>
<tr>
<td>Tracking</td>
<td>0.68</td>
</tr>
<tr>
<td>Saving (%)</td>
<td>80.8</td>
</tr>
<tr>
<td>Ratio(%)</td>
<td>5.16</td>
</tr>
</tbody>
</table>

Table 1. Computational saving of the surface tracker

---

\(^2\) We do not intend to compare the proposed method with the MC method, since it may be not appropriate. However, it is difficult to exactly re-implement other improved MC methods and to evaluate performance comparison on different platforms. Therefore, like previous studies [9,10], we use MC method as a golden standard to evaluate the proposed method.
Figure 11, 12, 13 and 14 show experimental results for Colon, CT head, Stomach and Cube datasets. In these figures, we represent different meta-surfaces with different colors to observe the growing results of the meta-surface. Table 2 shows various thresholds used in these four experiments. These thresholds are defined in Section 3.2 (growing criteria). Table 3 shows the timing breakdown of these experimental results. In this table, the time taken in building meta-surface includes both surface-tracking and growing timings. Compared with the original MC algorithm, the total cost of the proposed method is increased by about 36% for Colon, 65% for CT head, 26% for Stomach and 9% for Cube dataset, respectively. However, we can achieve very high decimation rates such as 86% in Colon, 79% in CT head, 69% in Stomach and 94% in Cube. Rendered results of Stomach dataset, for example, for non-decimation (Figure 13 (f)) and decimation models (Figure 13 (g)–(j)) are very similar in the visual quality. However, the number of triangles decimated is very high up to 69%. In Table 3, we also show the number of meta-surface and triangles generated. These numbers show a consistent trend: as specifying larger thresholds in Table 2, we can generate less number of meta-surface and triangles. For example, when the larger thresholds are used for Colon dataset (Figure 11 (c) and (d)), the decimating rate is obviously increased from 78% to 86%. Other datasets such as CT head (Figure 12(b)) and Stomach (Figure 13 (g)–(j)), have the same trend in decimating rate.

To demonstrate the ability of decimation, we use a volumetric Cube dataset to evaluate the proposed method, too. Figure 14 shows different representations of the Cube dataset. In this example, each face of cube is successfully grown into a single meta-surface, and each meta-surface is triangulated into two triangular surfaces. However, for vertices on or near to the edges (as indicated by a circle in Fig14-(c)), because of large difference in $\theta_v$, we need to increase $\theta_v$ up to 30° in our experiments. For the non-edge vertices, we just need to set a smaller $\theta_N$, say 1°. Then, we can significantly decimate them in this example.
Table 2. Various thresholds used in Colon (Fig 11), CT head (Fig 12), Stomach (Fig 13) and Cube (Fig 14) datasets.

<table>
<thead>
<tr>
<th></th>
<th>Plane Normal ($\theta_{N_c}$)</th>
<th>Vertex Normal ($\theta_{V_c}$)</th>
<th>Dist (pixel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig11-(c)</td>
<td>$11^0$</td>
<td>$15^0$</td>
<td>0.6</td>
</tr>
<tr>
<td>Fig11-(d)</td>
<td>$15^0$</td>
<td>$20^0$</td>
<td>1.6</td>
</tr>
<tr>
<td>Fig12-(b)</td>
<td>$10^0$</td>
<td>$15^0$</td>
<td>1.1</td>
</tr>
<tr>
<td>Fig13-(b)</td>
<td>$18^0$</td>
<td>$15^0$</td>
<td>0.4</td>
</tr>
<tr>
<td>Fig13-(c)</td>
<td>$25^0$</td>
<td>$30^0$</td>
<td>0.5</td>
</tr>
<tr>
<td>Fig13-(d)</td>
<td>$35^0$</td>
<td>$35^0$</td>
<td>0.5</td>
</tr>
<tr>
<td>Fig13-(e)</td>
<td>$35^0$</td>
<td>$35^0$</td>
<td>1.2</td>
</tr>
<tr>
<td>Fig14-(c)</td>
<td>$1^0$</td>
<td>$30^0$</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 3. Timing breakdown of experimental results for Colon (Figure 11), CT head (Figure 12), Stomach (Figure 13) and Cube (Figure 14) datasets.

<table>
<thead>
<tr>
<th></th>
<th>Building Meta-surface</th>
<th>Triangulation</th>
<th>Total Time</th>
<th>No. of Meta-surfaces</th>
<th>No. of Triangles</th>
<th>Decimation rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig11-(b)</td>
<td>--</td>
<td>--</td>
<td>3.55</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Fig11-(c)</td>
<td>3.42</td>
<td>1.82</td>
<td>5.24</td>
<td>482</td>
<td>5113</td>
<td>78.11</td>
</tr>
<tr>
<td>Fig11-(d)</td>
<td>3.13</td>
<td>1.73</td>
<td>4.86</td>
<td>282</td>
<td>3323</td>
<td>85.78</td>
</tr>
<tr>
<td>Fig12-(a)</td>
<td>--</td>
<td>--</td>
<td>12.34</td>
<td>--</td>
<td>96548</td>
<td>--</td>
</tr>
<tr>
<td>Fig12-(b)</td>
<td>11.21</td>
<td>9.21</td>
<td>20.42</td>
<td>9566</td>
<td>20124</td>
<td>79.51</td>
</tr>
<tr>
<td>Fig13-(a)</td>
<td>--</td>
<td>--</td>
<td>32.32</td>
<td>--</td>
<td>72652</td>
<td>--</td>
</tr>
<tr>
<td>Fig13-(b)</td>
<td>29.12</td>
<td>15.01</td>
<td>44.13</td>
<td>11146</td>
<td>36332</td>
<td>49.99</td>
</tr>
<tr>
<td>Fig13-(c)</td>
<td>28.01</td>
<td>14.87</td>
<td>42.88</td>
<td>7933</td>
<td>32704</td>
<td>54.98</td>
</tr>
<tr>
<td>Fig13-(d)</td>
<td>27.45</td>
<td>14.12</td>
<td>41.57</td>
<td>6446</td>
<td>27894</td>
<td>61.60</td>
</tr>
<tr>
<td>Fig13-(e)</td>
<td>26.85</td>
<td>13.85</td>
<td>40.70</td>
<td>5197</td>
<td>22613</td>
<td>68.87</td>
</tr>
<tr>
<td>Fig14-(a)</td>
<td>--</td>
<td>--</td>
<td>0.11</td>
<td>--</td>
<td>764</td>
<td>--</td>
</tr>
<tr>
<td>Fig14-(d)</td>
<td>0.10</td>
<td>0.02</td>
<td>0.12</td>
<td>26</td>
<td>44</td>
<td>94.25</td>
</tr>
</tbody>
</table>
Figure 11. CT Colon: (a) the original MC triangular mesh, (b) MC rendering results (c) Growing-cube, 78% decimation, (d) Growing-cube, 86% decimation, and (e) the \textit{meta-surface} of (d) case, where each meta-surface is shaded with different colors.

Figure 12. CT head: (a) the original MC rendered results and (b) at 80% decimating rate, rendered results using Growing-cube.
17

Figure 13. Stomach: (a) the original MC triangular mesh, (b)–(f) by the Growing-cube method at 49%, 54%, 61% and 68% decimating rates, respective and (f)–(j) the rendered results of (a)–(e), respectively.

Figure 14. Cube: (a) the original MC triangular mesh, (b) MC triangular wire-frame, (c) and (d) are wire-frames of meta-surface and triangular mesh at 94% decimating rate using Growing-cube method and (e) shows the same meta-surface but encoded in different colors.

Next, we would like to measure accuracy of surface representations generated by the proposed method. For this purpose, we compare the difference in term of a voxel width between surfaces generated by the Growing-cube (GC) and the original MC algorithm. For above four experiments, Figure 15 (I), (II), (III) and (IV) visualize this measurement using color code and this color-coded visualization can help us understand to qualify the error introduced by the proposed method in global fashion. In each color bar, the lower (red) means the higher error introduced and the higher (blue) means the lower error generated. In each figure, we also indicate the average error in term of the width of a voxel. From these figures and their average errors, we
can see the Growing cube algorithm can significantly decimate the number of triangles as well as can preserve good quality of surfaces. For example, in Figure 15 (I), as we decimate 86% of triangles in the original Colon triangles, the average error introduced by the Growing-cube (GC) is 0.185 voxel width. Among these four examples, the Cube case performs the best. The average error is zero, so the color code (IV) of Cube is all blue.

(II) Average error for CT Head example in Figure 12

(III) Average error for Stomach example in Figure 13

(IV) There is no error introduced by GC for Cube example (e) in Figure 14

Figure 15. The visualization of error introduced by the Growing cube (GC) method using color code for four experiments. In these figures, we also indicate the average error introduced in term of the width of a voxel.
5. Summary

In this paper, we present a Growing-cube algorithm to improve implementation of the MC algorithm. The proposed approach includes an adaptive decimation technique to generate various resolutions of models according to user-specified thresholds. It tracks connected surfaces instead of exhaustive searching iso-surfaces cell by cell. Therefore, it saves computation timing in identifying iso-surfaces. For our four test datasets, this saving varies from 30% to 80%. We conclude this gain is in inverse proportion to the ratio of non-empty cells over the entire volume cells. The lower ratio it is, the more efficiency it gains. The complete isosurface generated by the surface tracker is identical to that of MC method. Additionally, the proposed algorithm does not require additional data structure like Octree to speed up traversing cells [16] or to decimate triangular mesh [8,9]. From our experimental results, we find that the visual quality of rendering for both non-decimated and decimated models is nearly indistinguishable. For example, see rendered results for Stomach dataset. Therefore, compared with that of the MC method, we can use decimated models instead in our applications to save rendering time, too. Additionally, the total execution cost of the proposed algorithm is increased ranging from 9% to 65 % for all experiments, while we achieve decimating rate varying from 78% to 94%. From this respect, it is quite cost-effective. Some work can be done in near future. For example, in the current implementation, the proposed method grows face in a greedy manner. The greedy approach might be a sub-optimal method only. In future, we will develop a strategy to optimize our growing stage.

Acknowledgements

This work is supported in partial by NSC-89-2218-E-006-028, National Science Council, Taiwan, Republic of China

References

Dr. Tong-Yee Lee was born in Tainan county, Taiwan, Republic of China, in 1966. He received his B.S in computer engineering from Tatung Institute of Technology in Taipei, Taiwan, in 1988, his M. S. in computer engineering from National Taiwan University in 1990, and his Ph.D in computer engineering from Washington State University, Pullman, in May 1995. Now, he is an Associate Professor in the department of Computer Science and Information Engineering at National Cheng-Kung University in Tainan, Taiwan, Republic of China. He was with WSU as a Visiting Research Professor at School of EE/CS during 1996 summer and serves as guest associated editor for IEEE Transactions on Information Technology in Biomedicine in 2000. He has been working on parallel rendering and computer graphics since 1992, and has published more than 65 technical papers in referred journals and conferences. His current research interests include computer graphics, medical visualization, virtual reality, surgical simulation, distributed & collaborative virtual environment, parallel processing and heterogeneous computing.

Mr. Chao-Hung Lin was born in Koushung, Taiwan, Republic of China, in 1973. He received his B.S in computer science/engineering from Fu-Jen University and M.S in computer engineering from National Cheng-Kung University, Taiwan, Republic of China, in 1997 and 1998, respectively. Now, he is pursuing his Ph.D degree at Department of Computer Science and Information Engineering, National Cheng-Kung University. Mr. Lin research interests include computer graphics, image processing, virtual reality, visualization and interactive rendering.