A Marching Voxels Method for Surface Rendering of Volume Data[1]

Chin-Feng Lin, Don-Lin Yang, and Yeh-Ching Chung[2]
Department of Information Engineering,
Feng Chia University, Taichung, Taiwan 407
E-mail: {cflin, dlyang, ychung}@iecs.fcu.edu.tw

Abstract

The marching cubes method is a well-known surface extraction method by using the surface configurations of cubes for surface rendering of volume data. The marching cubes method has three main disadvantages, time consuming, ambiguity, and holes generation. All these disadvantages come from the use of the surface configurations of cubes. In this paper, we propose an efficient surface extraction method, the marching voxels method, for surface rendering of volume data. Instead of using the surface configurations of cubes, the marching voxels method first generates triangles for inner voxels. Then it combines the triangles of inner voxels to produce the surface of an object. Finally, the surface of an object is projected to a plane to form the final image. Since the marching voxels method considers the combination of triangles of voxels not cubes and the combination of triangles is performed in a deterministic way, there is neither ambiguous case of a combination nor holes for the generated surface. The experimental results show that the marching voxels method saves about 30% of the surface rendering time compared to the marching cubes method for test samples.

Index Terms: *Surface Rendering, Surface Extraction, Marching voxels Method, Marching Cubes Method.*

1. Introduction

In computer graphic, rendering is very important for volume visualization [5,23]. The surface rendering and the volume rendering are two major categories of rendering for volume visualization. For the surface rendering, it only shows the interested surface information of objects. In the volume rendering, it shows both inner and outer information of objects [4,6,26]. Since the surface rendering only handles the outer visible voxels of objects [2,8,15,16,18,22,24], it takes less rendering time than the volume rendering.

In the surface rendering, surface extraction is an important process to get visible and meaningful voxels from three-dimensional objects [1,3,9,11-14,17,19-20] and is a significant research subject. The marching cubes method proposed in [10] is a well-known surface extraction method by using the surface configurations of a cube for surface rendering of volume data. In the marching cubes method, a volume data is first partitioned into cubes. Each cube consists of eight voxels. Then it decides the surface configuration of each cube according to 15 surface configurations as shown in Figure 1. After determining the surface configuration of each cube, the surfaces of every two adjacent neighbor cubes are combined to form the surface of an object. The surface of an object is then projected to a plane to form the final image.

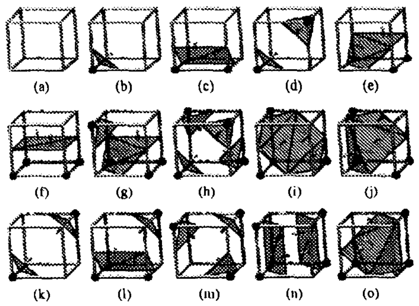


Figure 1. The 15 surface configurations of a cube.

The marching cubes method has several disadvantages. First, it takes a lot of time to perform surface rendering of large volume data. Second, during the surface combination of every two adjacent neighbor cubes, it may produce a wrong surface for the cubes due to more than one choice of the surface configuration of a cube. Third, it may produce some holes for the generated surface during the surface combination of every

[1] This work was partially supported by the NSC of ROC under contract NSC89-2213-E-035-032.
[2] The corresponding author.

two adjacent neighbor cubes. Some techniques have been proposed to solve the second and the third problems. For example, C. Montani *et al.* [11] and R. Seidel [21] proposed techniques to solve the ambiguous problem and the holes problem, respectively.

To overcome the disadvantages mentioned above, in this paper we propose an efficient surface extraction method, the marching voxels method, for surface rendering of volume data. Instead of generating the surface configurations of cubes, the marching voxels method first generates triangles for inner voxels. Then it combines the triangles of inner voxels to produce the surface of an object. Finally, the surface of an object is projected to a plane to form the final image. Since the marching voxels method considers the combination of triangles of voxels not cubes and the combination of triangles is performed in a deterministic way, there is neither ambiguous case of a combination nor holes for the generated surface. Therefore, the marching voxels method takes less time to extract surface of a volume data compared to the marching cubes method.

To evaluate the performance of the marching voxels method, we compare the proposed method along with the marching cubes method on an IBM RS6000 CPU. Three volume datasets are used as test samples. The experimental results show that the marching voxels method can save about 30% of the surface rendering time compared to the marching cubes method for test samples.

The rest of the paper is organized as follows. The marching voxels method will be described in Section 2. In Section 3, we compare the performance of the marching voxels method with the marching cubes method on an SP2 parallel machine and analyze the performance improvement of the marching voxels method.

2. The marching voxels method

As mentioned in the introduction section, the marching cubes method has three main disadvantages, time consuming, ambiguity, and holes generation. All these disadvantages come from the use of the surface configurations of cubes. In the marching voxels method, we consider the relations of surfaces of voxels. To generate the surface of a volume data, the marching voxels method consists three phases, the triangles generation phase, the polygons combination phase, and the surface projection phase. In the triangles generation phase, triangles are produced for inner voxels. In the polygons combination phase, polygons of voxels are combined according to the common vertices of polygons. In the surface projection phase, the surface of an object is projected to a plane to form the final image. Since the marching voxels method considers the combination of triangles of voxels not cubes, its execution is faster than the marching cubes method. In the marching voxels method, the combination of polygons is performed in a deterministic way. The ambiguity and holes generation problems of the marching cubes method are also eliminated. In the following, we describe the marching voxels method in detail and analyze the marching voxels method.

2.1 The algorithm of the marching voxels method

We first define some notations used in the marching voxels method.

<u>Definition 1</u>: A *volume data* $V = \{v_i = (x_i, y_i, z_i) \mid i = 1, 2, \ldots, n\}$ is defined as a finite set of voxels that form a cube, where (x_i, y_i, z_i) is the coordinate of v_i and $x_i, y_i, z_i \geq 0$.

<u>Definition 2:</u> An *object* $O = \{v_{in} \mid v_{in} \in V \text{ and } d(v_{in}) \geq D\}$ is defined as a set of voxels whose density $d(v_i)$ are greater than or equal to a threshold D. A voxel in O is called the *inner voxel*; otherwise the *outer voxel*.

<u>Definition 3:</u> Two voxels, $v_i = (x_i, y_i, z_i)$ and $v_j = (x_j, y_j, z_j)$, are neighbors if $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| = 1$. For voxel $v_i = (x_i, y_i, z_i)$, we define voxel $(x_i + 1, y_i, z_i)$ as its neighbor in direction x^+, voxel $(x_i - 1, y_i, z_i)$ as its neighbor in direction x^-, voxel $(x_i, y_i + 1, z_i)$ as its neighbor in direction y^+, voxel $(x_i, y_i - 1, z_i)$ as its neighbor in direction y^-, voxel $(x_i, y_i, z_i + 1)$ as its neighbor in direction z^+, and voxel $(x_i, y_i, z_i - 1)$ as its neighbor in direction z^-.

<u>Definition 4:</u> A voxel $v_i = (x_i, y_i, z_i)$ and its neighbors form eight quadrants. We define quadrants (x^+, y^+, z^+), (x^-, y^+, z^+), (x^-, y^-, z^+), (x^+, y^-, z^+), (x^+, y^-, z^-), (x^-, y^-, z^-), (x^-, y^+, z^-), and (x^+, y^+, z^-) as the first to the eighth quadrants of voxel $v_i = (x_i, y_i, z_i)$, respectively.

The first phase of the marching voxels method is to generate triangles for inner voxels. Since each inner voxel has eight quadrants, eight triangles are generated. Each triangle is intersected with the x-axis, y-axis, and z-axis. The intersection points of triangles of voxel v_i on $x^+, x^-, y^+, y^-, z^+, z^-$ are labeled as $v_i(x^+)$, $v_i(x^-)$, $v_i(y^+)$, $v_i(y^-)$, $v_i(z^+)$, and $v_i(z^-)$, respectively. The vertex of a triangle is in the middle of two adjacent voxels. An example is given in Figure 2.

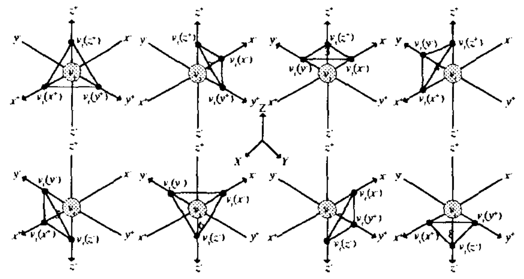
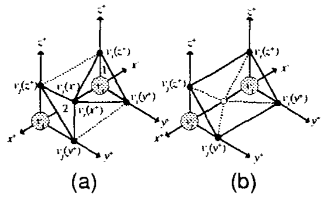


Figure 2. An example of eight triangles generated for an inner voxel.

The second phase of the marching voxels method is to combine polygons of voxels to form the surface of an object. It consists of four steps. In the first step, it tries to combine triangles of every two adjacent inner voxels on the x^+-axis starting from voxels in the yz plane with $x_i = 0$. We have the following case.

Case 1.1: Combine two triangles into a quadrangle. Given two adjacent inner voxels $v_i = (x_i, y_i, z_i)$ and $v_j = (x_i+1, y_i, z_i)$, the triangle in the first quadrant of v_i and the triangle in the second quadrant of v_j have a common vertex $v_i(x^+) = v_j(x^-)$ on the x-axis. To combine these two triangles, any two vertices that share edges with the common vertex $v_i(x^+)$ are jointed by an edge if they are in the same plane. After the combination, we can get a quadrangle. For example, Figure 3(a) shows two triangles of inner voxels v_i and v_j in the first quadrant and the second quadrant of v_i and v_j, respectively. Vertices $v_i(y^+)$ and $v_j(y^+)$ share edges with $v_i(x^+) = v_j(x^-)$ on the x-axis. Since $v_i(y^+)$ and $v_j(y^+)$ are in the same plane (the xy plane), edge $(v_i(y^+), v_j(y^+))$ is added. Edge $(v_i(z^+), v_j(z^+))$ is added as well with a similar reason. As a result, a quadrangle is formed and is shown in Figure 3(b).

We can get quadrangles in the forth, the fifth, and the eighth quadrants of voxel v_i in a similar manner.



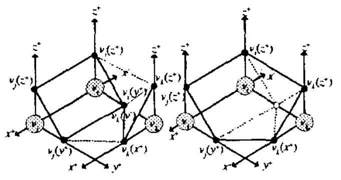
(a) (b)

Figure 3. An example of combining two triangles.

In the second step, it tries to combine triangles or quadrangles of every two adjacent inner voxels on the y^+-axis starting from voxels in the xz plane with $y_i = 0$. In the first step, two triangles may be combined into a quadrangle. To perform the combination process in this step, we have three cases. The first case is to combine two triangles into a quadrangle. The second case is to combine a triangle and a quadrangle into a pentagon. The third case is to combine two quadrangles into a quadrangle. Given two adjacent inner voxels $v_i = (x_i, y_i, z_i)$ and $v_j = (x_i, y_i+1, z_i)$, for these three cases, the combination of two polygons is similar to that of described in the first step, that is, any two vertices that share edges with the common vertex $v_i(y^+)$ are jointed by an edge if they are in the same plane. The combination is also applied to polygons in the second, the seventh, and the eighth quadrants of voxel v_i. We now describe the combinations of these three cases in detail.

Case 2.1: Combine two triangles into a quadrangle. This case is the same as Case 1.1.

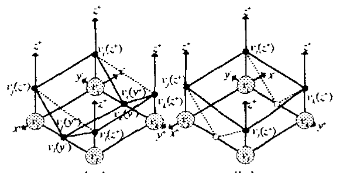
Case 2.2: Combine a triangle and a quadrangle into a pentagon. An example is shown in Figure 4. In Figure 4(a), for voxels v_i and v_k, the quadrangle of v_i and the triangle of v_k have a common vertex $v_i(y^+)$. Vertices $v_i(z^+)$ and $v_k(z^+)$ share edges with $v_i(y^+)$ on the y-axis. Since $v_i(z^+)$ and $v_k(z^+)$ are in the same plane (the yz plane), edge $(v_i(z^+), v_k(z^+))$ is added. Edge $(v_j(y^+), v_k(x^+))$ is added as well with a similar reason. As a result, a pentagon is formed and is shown in Figure 4(b).



(a) (b)

Figure 4. An example of combining a triangle and a quadrangle.

Case 2.3: Combine two quadrangles into a quadrangle. An example is shown in Figure 5. In Figure 5(a), for voxels v_i and v_k, the quadrangle of v_i and the quadrangle of v_k have two common vertices $v_i(y^+)$ and $v_j(y^+)$. Vertices $v_i(z^+)$ and $v_k(z^+)$ share edges with $v_i(y^+)$ on the y-axis. Since $v_i(z^+)$ and $v_k(z^+)$ are in the same plane (the yz plane), edge $(v_i(z^+), v_k(z^+))$ is added. Vertices $v_j(z^+)$ and $v_l(z^+)$ share edges with $v_j(y^+)$ on the y-axis. Since $v_j(z^+)$ and $v_l(z^+)$ are in the same plane (the yz plane), edge $(v_j(z^+), v_l(z^+))$ is added. As a result, a quadrangle is formed and is shown in Figure 5(b).



(a) (b)

Figure 5. An example of combining two quadrangles into a quadrangle.

In the third step, it tries to combine triangles, quadrangles, or pentagons of every two adjacent voxels on the z^+-axis starting from voxels in the xy plane with $z_i = 0$. The combination process is performed when two adjacent voxels are inner voxels. To perform the combination process in this step, we have seven cases: two triangles, a triangle and a quadrangle, a triangle and a pentagon, two quadrangles, a quadrangle and a pentagon, two pentagons, and a hexagon and a triangle. Given two adjacent inner voxels $v_i = (x_i, y_i, z_i)$ and $v_j = (x_i, y_i, z_i+1)$, for these seven cases, the combination of two polygons is similar to that of described in the first step, that is, any two vertices that share edges with the common vertex $v_i(z^+)$ are jointed by an edge if they are in the same plane. The combination is also applied to polygons in the second, the third, and the fourth quadrants of voxel v_i. We now describe the combinations of these seven cases in detail.

Case 3.1: Combine two triangles into a quadrangle. This case is the same as Case 1.1.

Case 3.2: Combine a triangle and a quadrangle into a pentagon. One example is the same as Case 2.2. Another example is shown in Figure 6. In Figure 6(a), for voxels v_i and v_m, the quadrangle of v_i and the triangle of v_m have a common vertex $v_i(z^+)$. Vertices $v_k(z^+)$ and $v_m(y^+)$ share edges with $v_i(z^+)$ on the z-axis. Since $v_k(z^+)$ and $v_m(y^+)$ are in the same plane (the yz plane), edge $(v_k(z^+), v_m(y^+))$ is added. Edge $(v_j(z^+), v_m(x^+))$ is added as well with a similar reason. As a result, a pentagon is formed and is shown in Figure 6(b).

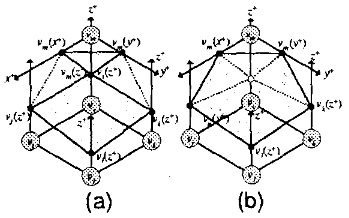

(a) (b)

Figure 6. An example of combining a triangle and a quadrangle into a pentagon.

Case 3.3: Combine a triangle and a pentagon into a hexagon. An example is shown in Figure 7. In Figure 7(a), for voxels v_i and v_l, the pentagon of v_i and the triangle of v_l have a common vertex $v_i(z^+)$. Vertices $v_j(z^+)$ and $v_l(x^+)$ share edges with $v_i(z^+)$ on the z-axis. Since $v_j(z^+)$ and $v_l(x^+)$ are in the same plane (the xz plane), edge $(v_j(z^+), v_l(x^+))$ is added. Edge $(v_k(z^+), v_l(y^+))$ is added as well with a similar reason. As a result, a hexagon is formed and is shown in Figure 7(b). In this case, we mark voxels that form the hexagon. In the given example, voxels v_i, v_j, v_k, and v_l are marked for further discussion later.

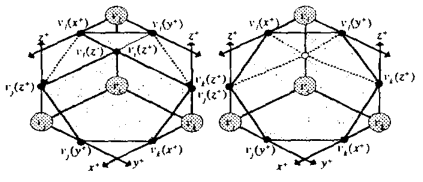

(a) (b)

Figure 7. An example of combining a triangle and a pentagon into a hexagon.

Case 3.4: Combine two quadrangles into a quadrangle or none. An example of combining two quadrangles into zero polygons is shown in Figure 8. In Figure 8(a), for voxels v_i and v_m, the quadrangle of v_i and the quadrangle of v_m have four common vertices $v_i(z^+)$, $v_j(z^+)$, $v_k(z^+)$, and $v_l(z^+)$. After the combination, all of the common vertices are disappeared. As a result, no polygon is formed as shown in Figure 8(b).

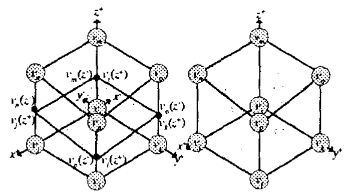

(a) (b)

Figure 8. An example of combining two quadrangles into zero polygon.

An example of combining two quadrangles into a quadrangle is shown in Figure 9. In Figure 9(a), for voxels v_i and v_m, the quadrangle of v_i and the quadrangle of v_m have a common vertex $v_i(z^+)$. Vertices $v_k(z^+)$ and $v_m(y^+)$ share edges with $v_i(z^+)$ on the z-axis. Since $v_k(z^+)$ and $v_m(y^+)$ are in the same plane (the yz plane), edge $(v_k(z^+), v_m(y^+))$ is added. Edge $(v_l(z^+), v_n(y^+))$ is added as well with a similar reason. As a result, a quadrangle is formed and is shown in Figure 9(b).

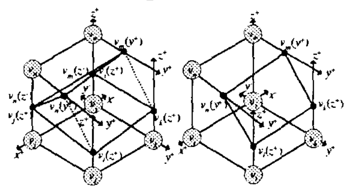

(a) (b)

Figure 9. An example of combining two quadrangles into a quadrangle.

Case 3.5: Combine a quadrangle and a pentagon into a triangle or a pentagon. An example of combining a quadrangle and a pentagon into a triangle is shown in Figure 10. In Figure 10(a), for voxels v_i and v_l, the pentagon of v_i and the quadrangle of v_l have a common vertex $v_k(z^+)$. Vertices $v_k(x^+)$ and $v_o(z^-)$ share edges with $v_k(z^+)$ on the z-axis. Since $v_k(x^+)$ and $v_o(z^-)$ are in the same plane (the xz plane), edge $(v_k(x^+), v_o(z^-))$ is added. Edge $(v_j(y^+), v_o(z^-))$ is added as well with a similar reason. As a result, a triangle is formed and is shown in Figure 10(b).

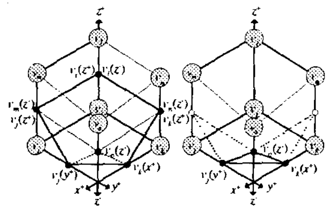
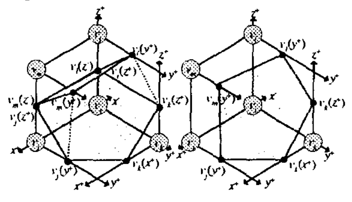

(a) (b)

Figure 10. An example of combining a pentagon and a quadrangle into a triangle.

An example of combining a pentagon and a quadrangle into a pentagon is shown in Figure 11. In Figure 11(a), for voxels v_i and v_l, the pentagon of v_i and the quadrangle of v_l have a common vertex $v_i(z^+)$.

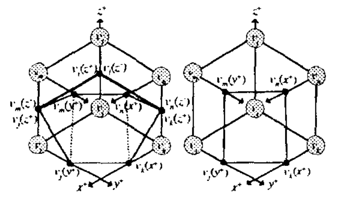
Vertices $v_k(z^+)$ and $v_l(y^+)$ share edges with $v_i(z^+)$ on the z-axis. Since $v_k(z^+)$ and $v_l(y^+)$ are in the same plane (the yz plane), edge $(v_k(z^+), v_l(y^+))$ is added. Edge $(v_j(y^+), v_m(y^+))$ is added as well with a similar reason. As a result, a pentagon is formed and is shown in Figure 11(b).



(a) (b)

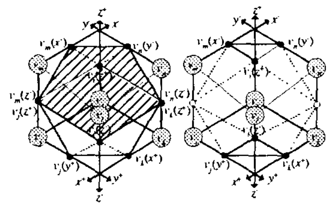
Figure 11. An example of combining a pentagon and a quadrangle into a pentagon.

Case 3.6: Combine two pentagons into a quadrangle or two triangles. An example of combining two pentagons into a quadrangle is shown in Figure 12. In Figure 12(a), for voxels v_j and v_l, the pentagon of v_j and the pentagon of v_l have a common vertex $v_k(z^+)$. Vertices $v_k(x^+)$ and $v_n(x^+)$ share edges with $v_k(z^+)$ on the z-axis. Since $v_k(x^+)$ and $v_n(x^+)$ are in the same plane (the xz plane), edge $(v_k(x^+), v_n(x^+))$ is added. Edge $(v_j(y^+), v_m(y^+))$ is added as well with a similar reason. As a result, a quadrangle is formed and is shown in Figure 12(b).



(a) (b)

Figure 12. An example of combining two pentagons into a quadrangle.

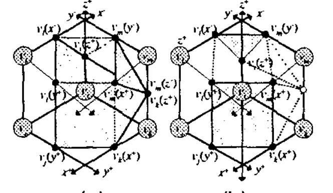


(a) (b)

Figure 13. An example of combining two pentagons into two triangles.

An example of combining two pentagons into two triangles is shown in Figure 13. In Figure 13(a), for voxels v_k and v_n, the pentagon of v_k and the pentagon of v_n have two common vertices $v_k(z^+)$ and $v_j(z^+)$. Vertices $v_i(z^+)$ and $v_n(y^-)$ share edges with $v_k(z^+)$ in the z-axis. Since $v_i(z^+)$ and $v_n(y^-)$ are in the same plane (the yz plane), edge $(v_i(z^+), v_n(y^-))$ is added. Edges $(v_k(x^+), v_l(z^-))$, $(v_i(z^+), v_m(x^-))$, and $(v_j(y^+), v_l(z^-))$ are added as well with a similar reason. As a result, two triangles are formed and are shown in Figure 13(b).

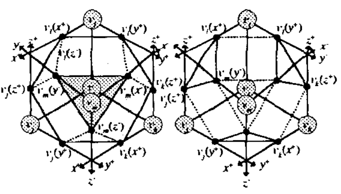
Case 3.7: Combine a hexagon and a triangle into a quadrangle and a triangle. An example is shown in Figure 14. In Figure 14(a), for voxels v_k and v_m, the pentagon of v_k and the triangle of v_m have a common vertex $v_k(z^+)$. Vertices $v_i(z^+)$ and $v_m(y^-)$ share edges with $v_k(z^+)$ on the z-axis. Since $v_i(z^+)$ and $v_m(y^-)$ are in the same plane (the yz plane), edge $(v_i(z^+), v_m(y^-))$ is added. The edge $(v_m(x^+), v_k(x^+))$ is added as well with a similar reason. Since vertices $v_m(x^+)$, $v_l(y^+)$, $v_l(x^-)$, and $v_m(y^-)$ form a quadrangle in a plane, a quadrangle and a triangle are formed and are shown in Figure 14(b).



(a) (b)

Figure 14. An example of combining a hexagon and a triangle into a triangle and a quadrangle.

The fourth step is to handle the exception. After Steps 1-3 are performed, a possible incorrect case shown in Figure 15(a) may be generated. In Figure 15(a), there are a hexagon and a triangle in the same cube. The correct surface configuration for the cube is shown in Figure 15(b). The following exception handling is required.



(a) (b)

Figure 15. An example of combining a hexagon and a triangle into three triangles.

Exception Case: For voxels marked in Case 3.3, if there exists a triangle in the same cube as the hexagon formed by the marked voxels, we combine the triangle and the hexagon into three triangles. In Figure 15(a), v_i, v_j, v_k, and v_l are marked voxels and v_m is the voxel for the triangle. Voxel v_i is adjacent to v_j, v_k, and v_l. To combine the hexagon and the triangle into three triangles, the vertex of the triangle on the x-axis is connected to the two vertices in the yz plane of voxel v_i. Similarly, the vertices of the triangle on the y-axis and z-axis are connected to the two vertices in the xz plane and in the xy plane of voxel v_i, respectively. In our example, vertex $v_m(x^-)$ is connected to $v_l(y^+)$ and $v_k(z^+)$ to form two edges $(v_m(x^-), v_l(y^+))$ and $(v_m(x^-), v_k(z^+))$. Edges $(v_m(y^-)$,

$v_l(x^+)$), ($v_m(y^-)$, $v_j(z^+)$), ($v_m(z^-)$, $v_j(y^+)$), and ($v_m(z^-)$, $v_k(x^+)$) are added as well with a similar reason. As a result, three triangles are formed and are shown in Figure 15(b).

The third phase of the marching voxels method is to project the surface of an object to an image plane. In this phase, we first use the linear interpolation method to compute the colors of surfaces and use the Phong shaded method [7] to produce smooth surfaces at the same time. Then the smooth surfaces are projected to an image plane and are displayed in a screen [25]. The algorithm of the marching voxels method is given below.

```
Algorithm Marching_Voxels_Method(V) {
/* V is the volume dataset. */
/* The triangles generation phase */
1.  Generate eight triangles for each inner voxel $v_i=(x_i, y_i, z_i)$ of the volume dataset V;
/* The polygons combination phase */
2.  For every inner voxel $v_i=(x_i, y_i, z_i)$, combine polygons of $v_i$ in the first, the forth, the fifth, and the eighth quadrants with its adjacent inner voxel $v_j=(x_i+1, y_i, z_i)$ according to Case 1.1;
3.  For every inner voxel $v_i=(x_i, y_i, z_i)$, combine polygons of $v_i$ in the first, the second, the seventh, and the eighth quadrants with its adjacent inner voxel $v_k=(x_i, y_i+1, z_i)$ according to Case 2.1 – Case 2.3;
4.  For every inner voxel $v_i=(x_i, y_i, z_i)$, combine polygons of $v_i$ in the first, the second, the third, and the fourth quadrants with its adjacent inner voxel $v_l=(x_i, y_i, z_i+1)$ according to Case 3.1 – Case 3.7;
5.  For the marked voxels in Case 3.3, perform the exception case;
/* The surface projection phase */
6.  Compute the color and shade the surfaces;
7.  Project the surfaces into an image plane;
}
end_of_Marching_Voxels_Method
```

2.2 The analysis of the marching voxels method

In order to compare the marching voxels method with the marching cubes method, we analyze the surface configurations generated by the marching voxels method in a cube according to the positions of inner voxels of the cube. Given a cube with eight voxels, we have nine combinations of inner voxels. Each combination may have more than one inner voxel configuration. For each inner voxel configuration, it may have more than one cases according to the positions of inner voxels. In the following, we give a general description for each configuration.

Combination 0: Zero inner voxel and eight outer voxels. We have one inner voxel configuration.

Combination 1: One inner voxel and seven outer voxels. We have one inner voxel configuration.

Combination 2: Two inner voxels and six outer voxels. We have two possible inner voxel configurations.

C2.1: Two inner voxels are adjacent.

C2.2: Two inner voxels are not adjacent.

Combination 3: Three inner voxels and five outer voxels. We have three inner voxel configurations.

C3.1: Three inner voxels are not adjacent.

C3.2: Only two inner voxels are adjacent.

C3.3: Three inner voxels are connected.

Combination 4: Four inner voxels and four outer voxels. We have five inner voxel configurations.

C4.1: Four inner voxels are not adjacent.

C4.2: Two pairs of inner voxels are adjacent.

C4.3: Only three inner voxels are connected.

C4.4: Four inner voxels are connected in a plane.

C4.5: Four inner voxels are connected but not in a plane.

Combination 5: Five inner voxels and three outer voxels. We have three inner voxel configurations.

C5.1: Three outer voxels are not adjacent.

C5.2: Only two outer voxels are adjacent.

C5.3: Three outer voxels are connected.

Combination 6: Six inner voxels and two outer voxels. We have two inner voxel configurations.

C6.1: Two outer voxels are adjacent.

C6.2: Two outer voxels are not adjacent.

Combination 7: Seven inner voxels and one outer voxel. We have one inner voxel configuration.

Combination 8: Eight inner voxels and zero outer voxel. We have one inner voxel configuration.

The surface configurations generated for the inner voxel configurations of these nine combinations by using the marching voxels method are listed in Table 1. In Table 1, the first column indicates the nine combinations of inner voxels. The second column lists the inner voxel configurations of the combinations of inner voxels. The third column lists the surface configuration generated by the marching voxels method for a given voxel configuration of a combination. The fourth column shows the generation steps of a surface configuration by the marching voxels method. From Table 1, we can see that all the surface configurations shown in Figure 1 can be generated by the marching voxels method. In addition, the surface configurations generated by the marching voxels method are performed in a deterministic way, that is, for a given case of inner voxel configuration of a cube, the surface configuration generated by the marching voxels method is unique. The ambiguity and holes generation problems of the marching cubes method are eliminated in the marching voxels method.

Table 1: The surface configurations generated by the marching voxels method.

Combi-nation	Inner Voxel Config-uration	The Surface Configuration Generated	The Generation Steps
0	–	Figure 1(a)	Phase 1
1	–	Figure 1(b)	Phase 1
2	C2.1	Figure 1(c)	Phase 1→Case 1.1
	C2.2	Figure 1(d)	Phase 1
	C2.2	Figure 1(k)	Phase 1
3	C3.1	Figure 1(m)	Phase 1
	C3.2	Figure 1(l)	Phase 1→Case 1.1
	C3.3	Figure 1(e)	Phase 1→Case 1.1→Case 2.2
4	C4.1	Figure 1(h)	Phase 1
	C4.2	Figure 1(n)	Phase 1→Case 1.1
	C4.3	Figure 1(g)	Phase 1→Case 1.1→Case 2.2
	C4.4	Figure 1(f)	Phase 1→Case 1.1→Case 2.3
	C4.5	Figure 1(i)	Phase 1→Case 1.1→Case 2.2→ Case 3.3
	C4.5	Figure 1(j)	Phase 1→Case 1.1→Case 2.2→ Case 3.3
	C4.5	Figure 1(o)	Phase 1→Case 1.1→Case 2.2→ Case 3.3
5	C5.1	Figure 1(m)	Phase 1→Case 1.1→Case 2.2→ Case 3.3→Exception Case
	C5.2	Figure 1(l)	Phase 1→Case 1.1→Case 2.2→ Case 3.3→Case 3.7
	C5.3	Figure 1(e)	Phase 1→Case 1.1→Case 2.3→ Case 3.2
6	C6.1	Figure 1(c)	Phase 1→Case 1.1→Case 2.2→ Case 3.6
	C6.2	Figure 1(d)	Phase 1→Case 1.1→Case 2.2→ Case 3.6
	C6.2	Figure 1(k)	Phase 1→Case 1.1→Case 2.2→ Case 3.6
7	–	Figure 1(b)	Phase 1→Case 1.1→Case 2.3→ Case 3.5
8	–	Figure 1(a)	Phase 1→Case 1.1→Case 2.3→ Case 3.4

3. Experimental results

To evaluate the performance of the marching voxels method, we have implemented the marching voxels method along with the marching cubes method on an IBM RS6000 CPU. Three volume datasets are used to evaluate the performance of the proposed surface rendering methods. They are selected from the Chapel Hill Volume Rendering Test Dataset. The first test sample is a "brain" dataset generated from the MR scan of a human head and the dimensions of the dataset is 128× 128 × 84. The second test sample is a CT "head" dataset and the dimensions of the dataset is 256 × 256 × 225. The third test sample is an "engine" dataset, which is the CT scan of an engine block and the dimensions of the dataset is 256× 256 × 110. Each image for the surface rendering methods is grayscale color and contains 512 × 512 pixels. Figure 16 shows the surface rendering time of the marching cubes and the marching voxels methods for the test samples "brain", "head", and "engine" on the IBM SP2 machine. In Figure 16, the performance improvement of the marching voxels method over the marching cubes method is also given. The performance improvement of the marching voxels method over the marching cubes method is defined as $1-\frac{Tv}{Tc}$, where T_c and T_v are the surface rendering time of the marching cubes method and the marching voxels method, respectively. From Figure 16, we can see that the surface rendering time of the marching voxels method is about 30% less than that of the marching cubes method for test samples.

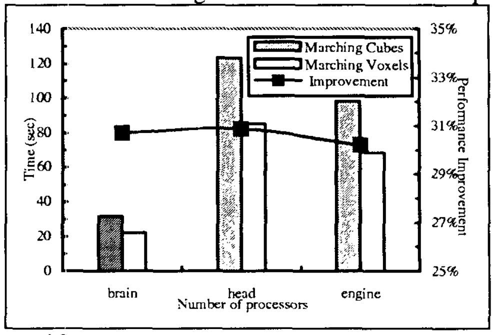


Figure 16: The total surface rendering time and the performance improvement of the marching voxels method over the marching cubes method for all test samples.

4. Conclusions

In this paper, we have proposed an efficient surface extraction method, the marching voxels method, for surface rendering of volume data. The marching voxels method first produces triangles for inner voxels. Then polygons of voxels are combined according to the common vertices of polygons. Finally, the surface of an object is projected to a plane to form the final image. Since the marching voxels method considers the combination of triangles of voxels and the combination of triangles is performed in a deterministic way, there is neither ambiguous case of a combination nor holes for the generated surface. To evaluate the performance of the proposed method, we have implemented the method on an IBM RS6000 CPU. Three volume datasets are used as test samples. The experimental results show that the marching voxels method saves about 30% of the surface rendering time compared to the marching cubes method for the test samples.

References

[1] F. Allamandri, P. Cignoni, C. Montani, and R. Scopigno, "Reconstruction of Topologically Correct and Adaptive Trilinear Iso-surfaces," *Computer & Graphics, Elsevier Science B. V.*, 1999 (in press).

[2] Ruud M.. Bolle and Baba C. Vemuri, "On Three-Dimensional Surface Reconstruction Methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 13, no 1, pp 1 - 13, January, 1991.

[3] P. Cignoni, C. Montani, E. Puppo and R. Scopigno, "Optimal Iso-surface Extraction from Irregular Volume Data," *IEEE/ACM 1996 Symp. on Volume Visualization*, S. Francisco CA, ACM Press, 1996, pp.31-38.

[4] Balázs Csébfalvi, "Fast Volume Rotation Using Binary Shear-Warp Factorization," *IEEE TCCG Symposium on Visualization*, Vienna, Austria, 1999.

[5] A. Kaufman (Eds.), Volume Visualization, *IEEE Computer Society Press*, 1991.

[6] Philippe Lacroute, "Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 3, pp. 218-231, 1996.

[7] Philippe Lacroute and Marc Lavoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the View Transformation," *Computer Graphics (SIGGRAPH '94 Proceedings)*, pp. 451-457, 1994.

[8] Marc Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May, 1988.

[9] Jianping Li;, Pen Agathoklis, "An Efficiency Enhanced Iso-surface Generation Algorithm for Volume Visualization," *The Visual Computer*, vol. 13, pp. 301-400, 1997.

[10] W.E. Lorensen, and H.E. Cline, "Marching Cubes: a high resolution 3D surface reconstruction algorithm," *Computer Graphics*, vol. 21, no. 4, pp 163-169 (Proc. of SIGGRAPH), 1987.

[11] C. Montani, R. Scateni, and R. Scopigno, "Decreasing Iso-surface Complexity via Discrete Fitting," *Computer-Aided Geometric Design, Elsevier Science*, 1998 (in press).

[12] C. Montani, R. Scateni, and R. Scopigno, "Discretized Marching Cubes," In R.D. Bergeron and A.E. Kaufman, editors, *Visualization '94 Proceedings*, pages 281-287. IEEE Computer Society Press, 1994.

[13] C. Montani, R. Scateni, and R. Scopigno, "A modified look-up table for implicit disambiguation of Marching Cubes," The Visual Computer, 10(6):353-355, 1994.

[14] C. Montani, and R. Scopigno, "Using Marching Cubes on small machines," *CVIGP: Graphical Models and Image Processing*, 56(2):182-183, March 1994.

[15] C. Oblonsek and N. Guid, "A Fast Surface-Based Procedure for Object Reconstruction from 3-D Scattered Points", *Computer Vision and Image Understanding*, vol. 69, no 2, pp 185 - 195, February, 1998.

[16] S. Parker, et al, "Interactive Ray Tracing for Iso-surface Rendering," in *Proceedings of Visualization '98*, pages 233-238.

[17] Tim Poston, Tien-Tsin Wong, Pheng-Ann Heng, "Multiresolution Iso-surface Extraction with Adaptive Skeleton Climbing," .

[18] Vaughan Pratt, "Direct Least-Squares Fitting of Algebraic Surfaces", *Proceedings SIGGRAPH 87 - Computer Graphics*, pp 145-152, 1987.

[19] Raj Shekhar, Elias Fayyad, Roni Yagel, and Fredrick Cronhill, "Octree-Based Decimation of Marching Cubes Surfaces".

[20] J. Wilhems and A. Van Gelder, "Octree for faster iso-surface generation," *ACM Transaction of Graphics*, pp201-227, vol. 11, 1992.

[21] R. Seidel. "A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and Tortriangulating Polygons," *Computational Geometry: Theory and Applications*, pp. 51-64, 1991.

[22] Marc Soucy and Denis Laurendeau, "A General Surface Approach to the Integration of a Set of Range Views", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 17, no 4, pp 334 - 358, April, 1995.

[23] L. Westover, "Footprint Evaluation for Volume Rendering", *Computer Graphics (In Proceedings of SIGGRAPH'90)*, vol.24, pp. 367-376, Dallas, 1990.

[24] J. Wilhems and A. Van Gelder, "Octree for faster iso-surface generation," *ACM Transaction of Graphics*, pp201-227, vol. 11, 1992.

[25] J. Wilhelms and A. Van Gelder, "A Coherent Projection Approach for Direct Volume Rendering," *Computer Graphics (In Proceedings of SIGGRAPH'91)*, vol. 25, no. 4, pp. 275-283, July 1991.

[26] C. Oblonsek and N. Guid, "A Fast Surface-Based Procedure for Object Reconstruction from 3-D Scattered Points", *Computer Vision and Image Understanding*, vol 69, no 2, pp 185 - 195, February, 1998.