Selectively Meshed Surface Representation

Chen Zhou, Renben Shu and Mohan S. Kankanhalli
Institute of Systems Science
National University of Singapore
Kent Ridge, Singapore 0511

Abstract

Surface rendering is an important technique for volume visualization. Any surface rendering algorithm has two phases - surface generation and rendering. We present a new surface rendering algorithm, which focuses on constructing the surface in a manner that speeds up the rendering phase. The motivation behind this is to reduce the response time for surface manipulations such as interactive rotations. We utilize a MC-like (Marching Cubes) approach to calculate the intersection points and their normals for each cube. But we dynamically link the intersection points to form triangles within the cube according to the locations of the last and the next visited neighboring cubes so that a good meshed surface can be generated. The difficulty with such an approach is that thousands of special cases need to be considered. But, we have found that the occurrence of 5 specific configurations out of the 14 basic MC cube configurations account for over 95% of all the cubes intersected by the iso-surface in most data sets. We process cubes belonging to these 5 configurations in a mesh mode, and the rest are processed in a non-mesh mode. As a result, the number of special cases are reduced substantially. Then a very careful analysis of the 5 configurations for mesh processing leads to just 136 cases, which makes the algorithm very simple. Test results show that the rendering time is almost halved compared to the time required for the rendering of a non-meshed surface generated by MC.

Key Words: Surface representation, Surface rendering, Mesh, Surface manipulation, Interactive Rotation.

Selectively Meshed Surface Representation

Chen Zhou, Renben Shu and Mohan S. Kankanhalli
Institute of Systems Science
National University of Singapore
Kent Ridge, Singapore 0511

1.0 Introduction

Surface rendering is a means of extracting meaningful and intuitive information from 3D data sets. It is achieved by converting the volume data into a surface representation using a surface extraction step (surface-generation phase) and then using conventional computer graphics techniques to render the surface (surface-rendering phase). The method of surface extraction is very important and several techniques have been developed to do this [5]:

- A method using a cuberille model, in which the rectilinear faces of all nontransparent voxels are used to form a polygon (square) represented surface [2, 4].

- A surface tracking algorithm that creates a surface from exterior voxel faces by starting from a seed on that surface and using a connectivity rule to form the rest of the surface [1, 4, 10, 11 and 12].

- The Dividing Cubes algorithm that generates a cloud of points [3].

- The Marching Cubes (MC) algorithm that creates a fine polygon (triangle) represented surface [7].

MC is the most popular surface generating algorithm since it is able to generate very high quality images by generating a set of triangles which closely approximates a surface of interest.

There are two phases in the process of displaying 3D surfaces reconstructed from a volume data set. One is the surface-generation phase, and the other is the surface-rendering phase. A surface with more than a million triangles is common when constructed from high resolution volume data set such as medical data. Even today a workstation with the best rendering performance has some difficulties in rendering a surface with over one million non-meshed triangles within three seconds, which is the normal response time expected by doctors in using an application such as surgical simulation. So it is necessary to speed up the rendering phase. This is the focus of this paper.

Basically, there are two approaches to speed up the rendering phase. One is to reduce the number of triangles needed to represent the same surface, and the other is to retain the same number of triangles, but use meshed triangles to represent the same surface. The rendering of meshed surface is more efficient than that of non-meshed surface because the shared vertices of two neighboring triangles can be rendered just once rather than three times for most vertices [9] when the triangles are treated as separate ones in a non-meshed mode. Rendering in a meshed mode is supported by certain industrial standards such as GL (Graphics Library) popularly used in Silicon Graphics workstations, IBM RS6000 workstations, etc. The former approach is more or less at the expense of image quality, whereas, for latter one, the original image quality can be preserved. And even if the former approach is applied, the latter one can still be used to mesh the surface, which could improve the speed further. We present a new surface rendering algorithm which outputs a meshed surface.

The remainder of this paper is organized as follows: Section 2 discusses general aspects of Selectively Meshed Surface Representation, including the definition of a mesh, analysis of another approach of meshed surface representation, the difficulties with meshed surface, and the basic SMSR idea. Then Section 3 describes the details of SMSR technique, including the framework of the method, and how to mesh cubes belonging to the five particular configurations in detail. After that Section 4 presents the results. And finally Section 5 concludes the paper with a discussion and suggestions for the future work.

2.0 Selectively Meshed Surface Representation (SMSR)

2.1 The Definition of Mesh

A set of triangles is called a triangular mesh or simply a mesh in this paper when they are defined by a specified sequence of triangle vertices and thus can be connectively rendered. Figure 1 shows an example of a simple mesh. In this example, the seven vertices form five triangles (123, 324, 345, 546, 567). Vertex 1 and vertex 7 appear in one triangle; vertices 2 and 6 appear in two triangles, and all the rest of the vertices each appear in three different triangles. In a larger mesh, a higher percentage of the points would be used three times. It is obvious that rendering the geometry in Figure 1 as a mesh is more efficient than rendering it as five separate triangles, because the mesh renders the shared vertices only once.

The specified sequence of vertices in the mesh in Figure 1 is as follows:

$$v_1, v_2, v_3, v_4, v_5, v_6, v_7.$$

This can be implemented quite easily using a FIFO queue which needs two memory locations to store the last two vertices. When a third vertex arrives, a new triangle is formed, and the first vertex moves out, whereas the third one moves in.

Figure 2 illustrates a more complex situation. The first six triangles (123, 324, 345, 546, 567, 768) could be rendered as before, but if nothing is done, the arrival of point 9 would cause triangle 789 to be rendered, not triangle 698 as desired. To render meshes like the one in Figure 2, the order of the two stored vertices must be exchanged. So 'S' (representing the operation for swapping the two vertices in the memory locations) should be added to the sequence of vertices when this situation occurs. The sequence of vertices for the mesh in Figure 2 is shown below:

$$v_1, v_2, v_3, v_4, v_5, v_6, v_7, S, v_8, S, v_9, S, v_{10}, v_4, v_{11}.$$

In real applications, meshes are not so simple and regular as those shown in Figure 1 and Figure 2. They are usually like the one shown in Figure 3.

Then a natural question is what constitutes a good mesh. We use the following criteria to define a good mesh:

(1) A good mesh should consist of as many triangles as possible. This minimizes the number of vertices rendered more than once.

(2) A good mesh should have the minimum number of swapping operations, if this is not at the expense of rule (1).

2.2 A Simple Meshed Surface Representation

It is possible to post-process non-meshed surface generated by MC, i.e. rearrange the sequence of the triangles so that they can be connectively rendered in the mesh mode. However, there are several disadvantages in doing this, shown as follows:

(a) Since it is very hard to give a guideline to select the next triangle from remaining two connecting ones if the current one is not the first triangle of a mesh, the whole surface would be randomly divided into patches. This is against rule (1) of a good mesh.

(b) The static formation of triangles in MC can not generate as a good mesh as the dynamic approach does.

In MC, the triangulation of a cube is static. By static, we mean that the triangulation of the vertices of a particular cube configuration is fixed, like Figure 4a which shows the fixed way of triangulating any cube in configuration 9 in MC. This is because the triangulated surface is in a non-mesh mode, and thus it is not necessary to consider how to triangulate a cube so that a good mesh can be formed. The pattern in Figure 4a could work well if the mesh in question is going to pass through the cube in Figure 4a, and the connected edge is p_1p_2. In such a case, a simple sequence of vertices p_1, p_2, p_6, p_3, p_5, p_4 can form a part of the mesh. Whereas if the connected edge is p_2p_3, then problems arise. First, no matter what choice we make, the mesh can not cover all these four triangles within a cube, which is against rule (1) of a good mesh. Even if we let the mesh cover triangles $p_2p_3p_6$, $p_3p_5p_6$, and $p_3p_4p_5$, there is still another problem. The sequence of vertices, p_2, p_3, p_6, S, p_5, p_4, is not a simple one. It is against rule (2) of a good mesh. The problem can be solved only if the triangles within a cube are formed dynamically, i.e. the vertices are linked into triangles based on the connected edge to triangulate a cube. Figure 4b illustrates a dynamic triangulation of the cube, given the connected edge p_2p_3. As can be seen, all the triangles within the cube are covered by one mesh and yet the sequence of vertices, p_2, p_3, p_1, p_4, p_6, p_5, is a simple one.

(c) Sorting non-meshed triangles (one million triangles is not an unusual case) to meshed ones consumes much more time than directly generating meshed triangles during the process of handling each cube.

Therefore, generating a meshed representation by post-processing is not acceptable. Thus, we now show SMSR, a method of directly producing a meshed representation.

2.3 Meshing Considerations

A lot of the difficulty with meshed surface representation method lies in the thousands of cases that have to be considered to form the meshed surface. We illustrate this by considering configuration 9 of MC in Figure 4a. In MC, for each configuration, formation of triangles within a cube is fixed in one way. Whereas, several factors need be considered to form meshed triangles. These factors are:

(1) *Location of edge*: Because in SMSR meshed triangles of current cube are formed dynamically based on locations of the last and the next visited cubes, locations of edges on the two cube faces shared by the last and the next cubes need to be considered. Thus, rotation symmetry used in MC is not applicable in SMSR, though complementary symmetry is applicable. This results in four cases derived from configuration 9. Figure 4a is one of the cases.

(2) *Input edge and output edge*: For any cube, if it is not a starting or ending cube of a mesh, there are two cubes called “last cube”, which has just been processed, and “next cube”, which will be processed immediately after the current one. The edge on the common face of the last and current cubes is called *an input edge*, and similarly, the edge on the common face of current and the next cubes is called *an output edge*. The input and output edges can be only on planes XZ or YZ (In our method, we limit a mesh to extend to another layer of cubes, that is why input and output edges can not be on plane XY). Because there are two faces on each of plane XZ or YZ, thus the number of choices to select both input and output edges is 12 (4 x 3).

(3) *Direction of input edge*: Even for given input and output edges, the direction of input edge should be considered, which results in two possible sequences of vertices. In Figure 4a, let’s assume that input edge is on the right face on plane YZ, and output edge is on the left face. If p_1p_2 is an input edge, the sequence for part of mesh is p_1, p_2, p_6, p_3, p_5, p_4. Whereas given p_2p_1 as the input edge, the sequence is p_2, p_1, p_3, p_6, p_4, p_5, which needs another formation of triangles.

Thus, for configuration 9 the total possible number of sequence of vertices is 96 (4 x 12 x 2) in SMSR.

There are 14 basic configurations in MC [7] shown in Figure 5. In order to solve the hole problem with the original MC, subconfigurations for 6 of the basic configurations are introduced in a more recent paper [13]. Specifically, there are 2 subconfigurations for configuration 3, 2 for configuration 6, 8 for configuration 7, 4 for configuration 10, 4 for configuration 12, and 64 for configuration 13. This results in a total number of 92 extended configurations. Recall that the number of sequence of vertices for configuration 9 is 96 whose complexity is no more than average. Thus the total number of cases for the extended configurations could be thousands, or may be even tens of thousands.

2.4 Selective Meshing

Fortunately, we have found that more than 99% of the cubes with the iso-surface crossing in a volume data set belong to five basic configurations 1, 2, 5, 8 and 9 (See Table 1.). Frequency of configuration has also been studied for other purposes by Wilhelms and Van Gelder [13], and Nielson and Hamann [8]. The experimental results from 4 volume data sets by Wilhelms and Van Gelder show that the occurrence of the 5 configurations accounts for around 95%, whereas the results from 4 volume data sets by Nielson and Hamann indicate that the figure is over 99%. This fact suggests that, the whole surface does not have to be fully meshed. Thus we can process cubes

belonging to only these 5 configurations in a mesh mode, and the others in a non-mesh mode. We call a cube belonging to one of the five configurations a *mesh cube*, otherwise a *non-mesh cube*.

The basic idea of SMSR is as follows:

(1) For a mesh cube, the processing result is a sequence of vertices for part of a mesh.

(2) For a non-mesh cube, we use exactly the same method as in MC to process it. The processing result is several independent triangles. Although we can let these independent triangles be a complete mesh or several complete meshes (if there are separated surfaces across the cube), the overall performance is almost the same because the number of these kinds of meshes is very small, and also the average length of these meshes is very small which can not show the advantage of mesh.

3.0 The SMSR Technique

The overall computation of SMSR is as follows:

(1) The sequence of cube processing is layer by layer in the direction of Z axis.

(2) Within each layer, the sequence of cube processing is determined by certain rules. These rules make sure that the triangulated surfaces in two consecutively processed cubes are connected, and the expanding trend of a mesh is regular so that random connection of triangles, which may result in surface being divided into too small patches, can be avoided. The scheduling of cubes is controlled by a Cube Manager, which will be discussed in Section 3.1.

(3) For each cube in question, according to the relative locations of the last and the next visited neighboring cubes, and the cube configuration, a specific triangulation is performed on the cube. This is done by a Triangle Manager, which will be discussed in Section 3.2.

One thing worth mentioning is the data structure for the storage of meshed surface. A large number of vertices are shared by either neighboring meshes or non-meshed triangles. If these shared vertices are stored several times, several megabytes memory would be wasted. Therefore a special data structure for non-redundant storage of vertices is needed.

Two tables are used for the non-redundant storage of vertices. One table, *vertex table*, registers all the coordinates and normals of vertices. The other table, *mesh table*, registers all the meshes, each vertex of which points to one entry of the vertex table. In addition, another auxiliary working table, whose structure is exactly the same as that of a layer of cubes, is needed to register all the vertices

so that the mesh table could be established. In SMSR, the sequence of cube processing is layer by layer in direction of Z axis. Within each layer, the sequence is unknown, which is dependent on the specific shape of the surface.

For processing any of the 12 edges, this needs to be done:

(1) For any edge among the four edges having been processed on the last layer, just fill in the reference table by acquiring the vertex index from working table.

(2) For any edge among the other eight edges, first consult the working table to see if the edge has been interpolated. If yes, do exactly the same thing as (1), else do the interpolation, and then register this newly interpolated vertex to the vertex table, fill in the working table and reference table with the returned vertex index from filling in vertex table.

3.1 The Cube Manager

The cube manager mainly deals with scheduling of cubes within one layer. The cube face notation is shown in Figure 6. Face $v_1v_5v_8v_4$ is denoted by -X, $v_2v_6v_7v_3$ by +X, $v_1v_2v_6v_5$ by -Y, and $v_4v_3v_7v_8$ by +Y. X (Y) denotes either -X or +X (either -Y or +Y) depending on the specific situation. These four faces could be either an input face or an output face. The *input face* is the face shared by current cube and the last cube if current cube is not a starting cube of a mesh, otherwise a default face is given for the input face, which is selected on the basis of no swapping operation in the sequence of vertices. The *output face* is the face shared by current cube and the next cube, if current cube is not an ending cube of a mesh, otherwise a default face is given for the output face on the same basis as that of the default input face. There is also a flag to denote whether the mesh is complete or not.

The framework of cube manager is as follows:

- Before processing cubes on each layer, we put all the cubes with a surface crossing in that layer into a set. Those cubes are physically allocated in a scan-line order with direction of X axis having a higher priority.

- Any cube being processed belongs to one of the following four states:

(1) The current cube is a mesh cube and current mesh is uncompleted.

Based on the input face and output face, the triangle manager meshes the current cube in a mesh mode. The output face is selected according to the priority order -X, +X, -Y, +Y (input and output faces cannot be the same) and simultaneously satisfying the condition that the neighboring cube sharing the output face is not yet processed. If all the neighboring cubes are processed, just select a default output face and turn the uncompleted mesh flag to FALSE.

(2) The current cube is a mesh cube and current mesh is completed.

We select it from the first element (physically) in the set. The output face is selected on the same basis as that in state (1). The input face is determined by the output face to try to make a segment of mesh without any swapping operation in it. Then the triangle manager meshes the current cube in a non-mesh mode. If there exists one non-processed neighboring cube, the uncompleted mesh flag is turned TRUE.

(3) The current cube is a non-mesh cube and current mesh is uncompleted.

End the uncompleted mesh and turn the uncompleted mesh flag to FALSE. The triangle manager triangulates the current cube in a non-mesh mode.

(4) The current cube is a non-mesh cube and current mesh is completed.

We select it from the first element (physically) in the set. The triangle manager triangulates the current cube in a non-mesh mode.

- A cube is moved out of the set after it has been processed.
- The processing of cubes on a layer halts after the set is empty.

3.2 The Triangle Manager

The triangle manager mainly deals with meshing cubes with those five configurations into meshed triangles, i.e. sequences of vertices (interpolated points). In addition, triangle manager is responsible for triangulating cubes with the other configurations in the same way as what MC does. So we shall discuss how to mesh cubes with only those five meshed cube configurations.

As we have analyzed in section 2.3, there are 96 cases (sequences of vertices) for configuration 9. If we calculate the cases in a similar way, the total number of cases for those five configurations could be several hundreds. For each configuration, a large number of cases is contributed by the product of the three factors mentioned in section 2.3. Although nothing can be done about the last

two factors, reducing the size of the first factor is possible. That is our focus for each of the five configurations.

The basic idea is like this: For each configuration, we partition all the cube cases into some groups based on locations of edges on cube faces, i.e. for any instance in the group the number of edges on each coordinate plane are same, and then partition each group further into some subgroups based on the meaningful formation of triangles. The purpose is to ensure that all the elements in the same subgroup can be processed in the same or similar way. We try to keep the number of subgroups as small as possible, since this finally will affect the number of sequences of vertices. A proper edge numbering is essential. This is because it determines whether some instances can be grouped in the same subgroup or they should be separated to different subgroups, which may cause an increase in the number of subgroups. For configurations 1, 2, and 8, this is straightforward, whereas for configurations 5 and 9, it is not.

For each configuration, we start with a figure showing all the representatives taken from each subgroups, and then give the sequence of vertices indexed by the input & output faces with the corresponding representative number in the figure. For each representative, there are several sequence of vertices corresponding to it.

We now show how to triangulate any instance which belongs to the five mesh cube configurations.

- *Configuration 1*

There is only one representative, shown in Figure 7, corresponding to the four cases of sequence of vertices. No matter what the instance is, the three edges of triangle on face X, Y and Z are always numbered exactly in the same manner, such as 23, 13, and 12 respectively here. This ensures that the treatment of any other instance of configuration 1 is exactly the same as that of the representative. Table 2 shows the sequence of vertices indexed by input & output faces in a bracket pair. Note the conventions are that X denotes either -X or +X, and it is similar to Y. The content of I/O face on even rows is the same as that on the immediate upper row, though we leave them blank. The first two vertices of the sequence are put in parentheses, because they actually belong to the last visited cube.

- *Configuration 2*

Six representatives are illustrated in Figure 8. They belong to three groups. Group I consists of (a) and (b), group II consists of (c) and (d), and group III consists of (c) and (d). Any instance whose

edge with the two marked vertices is parallel to the X axis, belongs to group I. Similarly, any instance whose edge with the two marked vertices is parallel to the Y axis belongs to group II, and any instance whose edge with the two marked vertices is parallel to the Z axis belongs to group III. Once the group is determined, the sequence of vertices can be selected based on the input & output faces. For any instance in group I, four edges on cube faces are always on faces -X, +X, Y, and Z, which are numbered as 12, 34, 24, and 13 respectively. Similarly, for any instance in group II, four edges on cube faces are always on faces -Y, +Y, X, and Z, which are numbered as 12, 34, 24, and 13 respectively. For any instance in group III, four edges on cube faces are always on faces -Z, +Z, X, and Y, which are numbered as 12, 34, 24, and 13 respectively. This is to make sure that sequences of vertices can be directly used. See Table 3 for sequence of vertices in detail.

- *Configuration 5*

Eleven representatives come from three groups that are classified based on the plane on which the three marked vertices lie. See Figure 9. Group I is made up of (a), (b), and (c). For any instance belonging to this group, five edges on cube faces are always on faces X, -Y, +Y, -Z, and +Z respectively. Similarly, Group II is made up of (d), (e), and (f), and five edges on cube faces are always on faces, X, +X, Y,-Z, and +Z respectively. Group III is made up of (g), (h), (i), (j) and (k), and five edges on cube faces are always on faces -X, +X, -Y, +Y, and Z respectively.

Unlike configurations 1 and 2, which were discussed previously, we have some difficulties in numbering the vertices. Let’s take a look at how to number the vertices for (a). Although we know that five edges on cube faces are always on faces X, -Y, +Y, -Z, and +Z respectively, there is no way to fix the numbering of edges on certain face for different instances, i.e. if given another instance falling into subgroup represented by (a), the numbering convention for (a) is not applicable to it.

Further investigation shows that although fixing the numbering of edges based on face is impossible, fixing based on coordinate plane is possible. For instance, in our numbering, for any instance falling into group I, the edge on face X is always numbered as 45, the edges on face Y are always numbered as 14 and 23, (though it is unknown which edge is on face -Y and which is on +Y,) and the edges on Z are always numbered as 12 and 35. Without this numbering method, the subgroups have to be divided again which would dramatically increase the number of sequences of vertices. The numbering convention for any instance in the other two groups are as follows: For any instance in group II, the edge on face Y is always numbered as 45, the edges on face X are always numbered as 14 and 23, and the edges on Z are always numbered as 12 and 35. For any instance in

group III, the edge on face Z is always numbered as 45, the edges on face X are always numbered as 14 and 23, and the edges on Y are always numbered as 12 and 35.

Like Table 2 and Table 3, Table 4 also shows all the sequences of vertices for all the representatives. But Table 4 can not be directly used as Table 2 and Table3, because the edges cannot be fixed on a certain face. However, it can be used on a reference basis and it is very easy to refer to it. For example, if given an instance belonging to group III, input edge as 23 (directed) and output face as -Y, by checking whether 12 or 35 is on -Y, either the sequence of vertices (23)S5S4S1 or (23)1S4S5 is selected.

- *Configuration 8*

Six representatives in Figure 10 belongs to three groups with (a) and (b) in group I, (c) and (d) in group II, and (e) and (f) in group III. Any instance in group I has four edges on -Y, +Y, -Z, and +Z. Whereas any instance in group II has four edges on -X, +X, -Z, and +Z, and any instance in group III has four edges on -X, +X, -Y, and +Y. Also numbering convention is clear. Table 5 shows the sequence of vertices.

- *Configuration 9*

There are six representatives shown in Figure 11. A similar numbering problem to that in configuration 5 occurs here. So similar measures to those in configuration 5 are taken to resolve the problem. As a result for any instance, edges on face X are always numbered as 12 and 45, edges on face Y are always numbered as16 and 34, and edges on Z are always numbered as 23 and 56. Like Table 4 for configuration 5, Table 6 is used mainly for illustration on a reference basis.

As we can see, there are 4 cases of sequence of vertices for configuration 1, 28 cases for configuration 2, 48 cases for configuration 5, 32 cases for configuration 8, and 24 cases for configuration 9, which results in a total of 136 cases.

4.0 Results

We implemented both the original MC and the SMSR algorithms on a Silicon Graphics workstation IRIS Indigo Elan 4000 and tested the implementation using three CT-scan data of human head. The rendering speed of this workstation is 225k meshed triangles per second.

We concentrated on measuring the time required for the rendering phase, assuming that the surface has already been generated and stored in primary memory. The test results shows that when we

render stored non-meshed surface generated by MC, the performance we have achieved is 83k triangles per second, 37% of manufacturer's data, which does not vary with different data sets. While when meshed surface generated by RMSR is rendered, we can achieve up to 161k meshed triangles per second, 71% of ideal figure. In short, the rendering speed is doubled.

The rendering speeds are the same for non-meshed surfaces generated by MC from different data sets, because all the triangle are independent. While the rendering speeds for meshed surface generated by SMSR from different data sets are different shown in our test results, because triangles are rendered connectedly in a mesh mode. The performance is determined by the length of meshes and the number of swapping operations, which vary from data set to data set. See Table 7.

One reason which prevents the implementation from achieving the ideal figure of 225k meshed triangles per second is the irregularity of triangulated surface. Even a small patch of surface needs many meshes to cover. Secondly, the million triangles cannot be held in the main memory which leads to a lot of virtual memory swapping. This also causes a performance degradation.

5.0 Conclusion and Discussions

The surface rendering process for visualization has two phases, namely the surface-generation phase and the surface-rendering phase. We have presented a new surface construction algorithm which builds a surface so that it can be rendered more efficiently. We achieve this by constructing a selectively meshed surface representation. The rendering of meshed surface is more efficient because the shared vertices of two neighboring triangles can be rendered just once rather than three times for most vertices of triangles treated as separate ones in a non-meshed mode.

Although there are other approaches to speed up rendering phase such as reducing the number of triangles to represent the same surface, they are usually at the expense of image quality. Whereas, for SMSR, the number of triangles is exactly the same with the number of those generated by MC to represent the same surface only with the exception of triangle shapes, which do not affect the image quality at all.

One could use a simple approach to post-process the independent triangles generated by MC, namely rearrange the sequence of all the triangles, so that the surface can be meshed. However, no efficient heuristics for this are available and consequently it results in the whole surface being divided into too small patches. In addition, sorting millions of triangles is quite expensive. Because the surface is selectively meshed, a natural question would be whether this method is efficient or not compared to the fully meshed approach. Calculations based on frequency distribution

and triangle number distribution among the 14 basic configurations of MC for various data sets show that meshed triangles account for 99% of all the triangles. So SMSR is indeed efficient for real data.

Another advantage of meshed surface representation, compared to the non-meshed surface representation, is that the number of references to the vertices are dramatically reduced, which saves a lot of memory storage space.

This meshed surface representation algorithm can be further improved. One could obtain a better quality meshed surface representation through a more intelligent strategy when we select the next visited cube. If the selection can be based on more cubes, say a small region of cubes, definitely the quality of meshed surface could be improved, i.e. longer meshes and smaller swap instructions will result at the cost of increased search time. However, the main obstacle in improving the quality of meshed surface is the irregularity of triangulated surface generated from real application data sets such as medical data. As shown in Figure 3, even that small patch needs at least three meshes to cover it. This aspect can be studied further for designing better algorithms.

References

1. E. Artzy, G. Frieder and G.T. Herman, The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. *Computer Graphics and Image Processing*, 15(1):1-24 (1981).

2. L. Chen, G.T. Herman, R.A. Reynolds and J.K. Udupa, Surface shading in the cuberille environment. *IEEE Computer Graphics and Applications*, **5**(12):33-43 (1985).

3. H.E. Cline, W.E. Lorensen, S. Ludke, C.R. Crawford and B.C. Teeter, Two algorithms for the three-dimensional reconstruction of tomograms. *Medical Physics*, **15**(3):320-327 (1988).

4. B. Gordon and J.K. Udupa, Fast surface tracking in three-dimensional binary images. *Computer Vision, Graphics and Image Processing*, 29(2):196-214 (1989).

5. G.T. Herman and H.K. Liu, Three-dimensional display of human organs from computed tomograms. *Computer Graphics and Image Processing*, 9(1):1-21 (1979).

6. A. Kaufman, Chapter 1: Introduction to volume visualization. *Volume Visualization.* IEEE Computer Society Press, Los Alamitos, California (1990).

7. W.E. Lorensen and H.E. Cline, Marching Cubes: a high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163-169 (1987).

8. G.M. Nielson and B. Hamann, The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes, *Proceedings of Visualization '91*, San Diego, CA, pp. 83-91 (1991).

9. Silicon Graphics Inc., Graphics Library Programming Guide, Mountain View, CA. (1991)

10. L. Sobierajski, A. Kaufman, D. Cohen, R. Yagel and D. Acker, Interactive volume visualization system for craniofacial surgery. *Technical Report 90.04.30*, Computer Science, State University of New York, Stony Brook, N.Y (1990).

11. S.S. Trivedi, G.T. Herman and J.K. Udupa, Segmentation into three classes using gradients. *IEEE Transactions on Medical Imaging*, MI-5(2):116-119 (1986).

12. J.K. Udupa and H.M. Hung, Surface versus volume rendering: a comparative assessment. In *Proceedings of the First Conference on Visualization in Biomedical Computing*, pp. 83-91, IEEE Computer Society Press, Los Alamitos, California (1990).

13. J. Wilhelms and A. Van Gelder, Topological Considerations in Isosurface Generation,*Computer Graphics*, 24(5):79-86 (1990).

14. J. Wilhelms and A. Van Gelder, Octree for Faster Isosurface Generation, *ACM Transaction on Graphics*, 11(3):201-227 (1992).

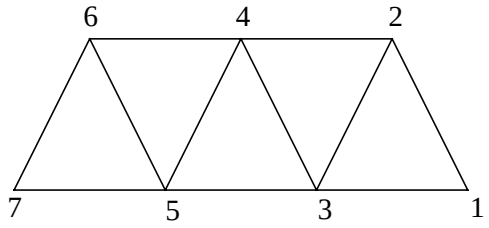

Figure 1. Example of a simple mesh

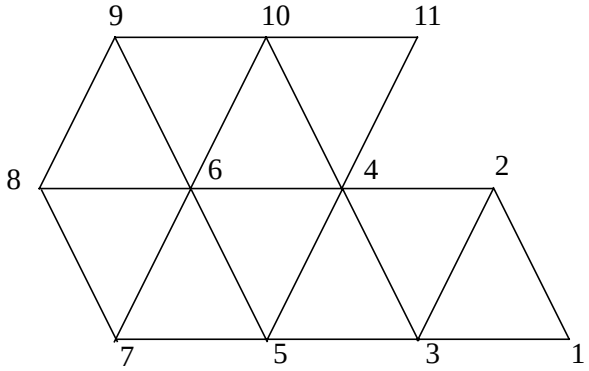

Figure 2 Example of a mesh requiring the swapping operation

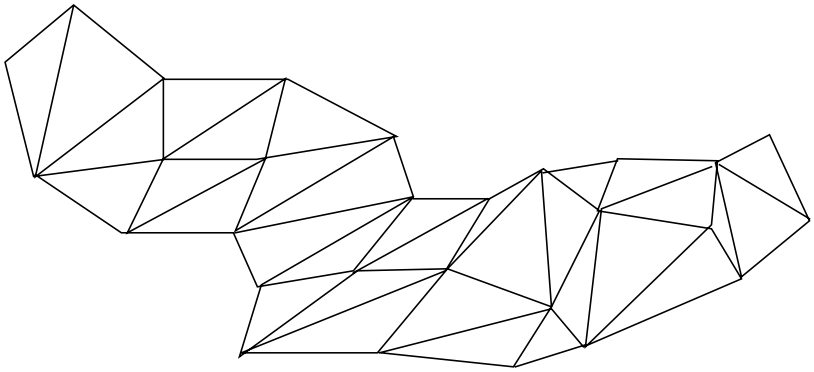

Figure 3. A mesh from a real application

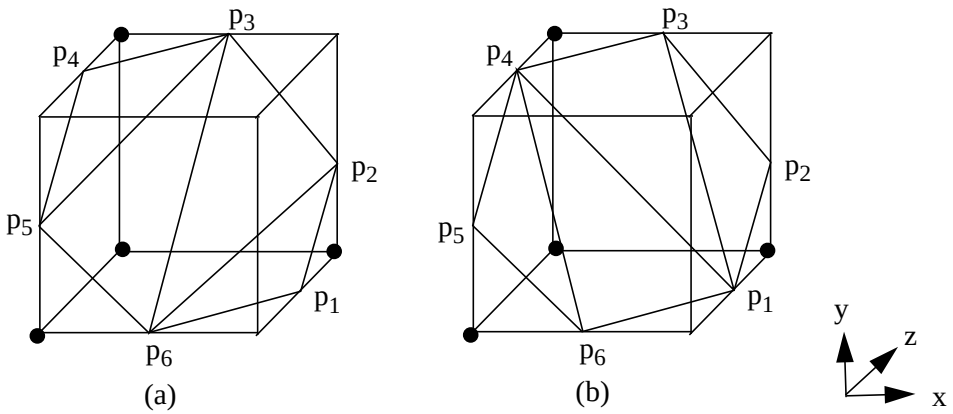

Figure 4. Static vs. dynamic formation of triangles within a cube.
(a) Static one (b) Dynamic one

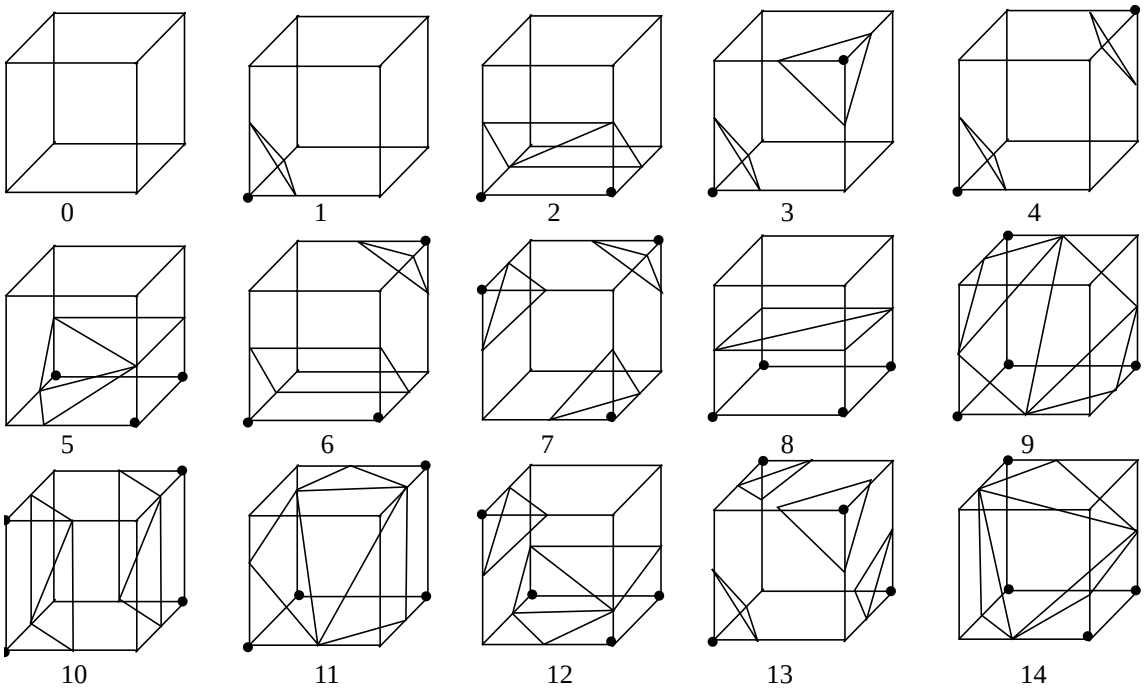

Figure 5. Configurations of triangulated cubes.

Data Set	CThead1	CThead2	CThead3
Configuration 1 :	13.24%	10.06%	10.23%
Configuration 2 :	16.97%	19.09%	13.80%
Configuration 3 :	0.30%	0.05%	0.06%
Configuration 4 :	0.05%	0.00%	0.05%
Configuration 5 :	9.34%	8.92%	9.64%
Configuration 6 :	0.14%	0.08%	0.16%
Configuration 7 :	0.01%	0.0%	0.00%
Configuration 8 :	59.14%	61.26%	65.62%
Configuration 9 :	0.69%	0.46%	0.23%
Configuration 10:	0.02%	0.03%	0.04%
Configuration 11:	0.03%	0.01%	0.06%
Configuration 12:	0.03%	0.01%	0.04%
Configuration 13:	0.00%	0.00%	0.0%
Configuration 14:	0.03%	0.02%	0.05%
Top 5 configurations	99.4%	99.8%	99.5%

Table 1. Configuration Frequency in CT-Scan Data of Human Head .

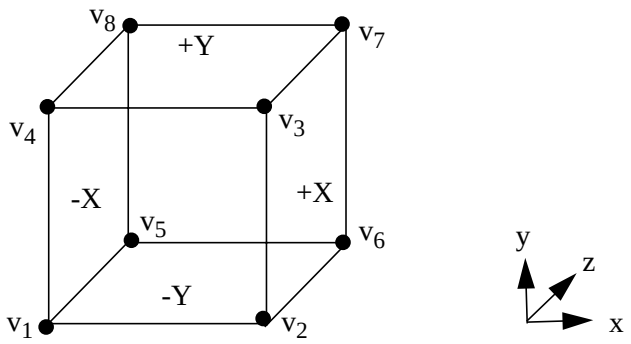

Figure 6. Cube face notation

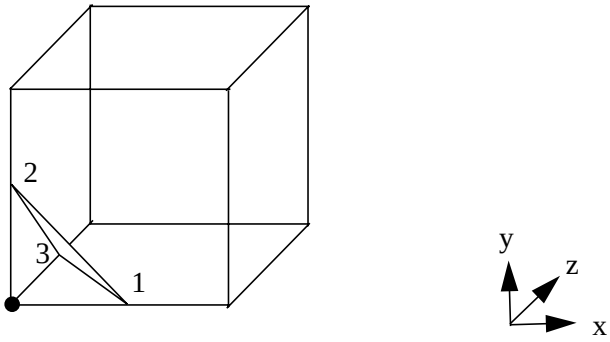

Figure 7. Representatives of Configuration 1

I/O Face	Sequence of Vertices
[X, Y]	(23)1 (32)S1
[Y, X]	(13)2 (31)S2

Table 2. Sequences of vertices for configuration 1

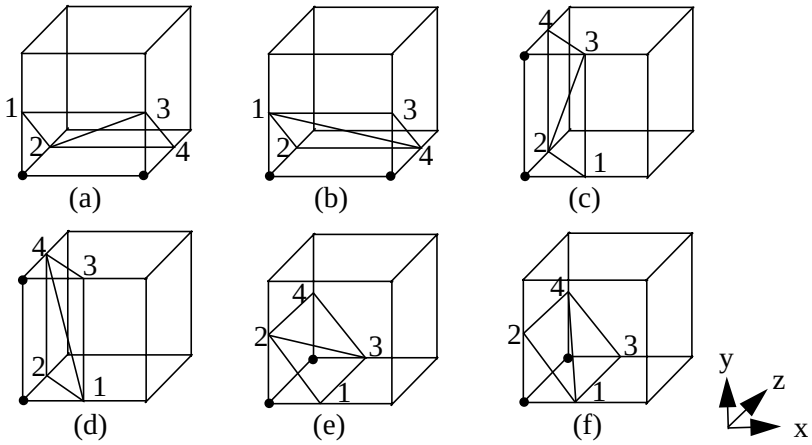

Figure 8. Representatives of Configuration 2

I/O Face	Representative	Sequence of Vertices
I [-X,+X]	a	(12)34
	b	(21)43
[-X, Y]	a	(12)3S4
	a	(21)S3S4
[+X,-X]	b	(34)12
	a	(21)S3S4
[+X, Y]	b	(34)1S2
	b	(43)S1S2
[Y, -X]	a	(24)S3S1
	a	(42)3S1
[Y,+X]	b	(24)1S3
	b	(42)S1S3
II [X,-Y]	c	(24)S3S1
	c	(42)3S1
[X,+Y]	d	(24)1S3
	d	(42)S1S3
[-Y, X]	c	(12)3S4
	c	(21)S3S4
[-Y,+Y]	c	(12)34
	d	(21)43
[+Y, X]	d	(34)1S2
	d	(43)S1S2
[+Y,-Y]	c	(34)12
	d	(43)21
III [X, Y]	e	(24)13
	f	(42)31
[Y, X]	f	(13)24
	e	(31)42

Table 3. Sequences of vertices for configuration 2

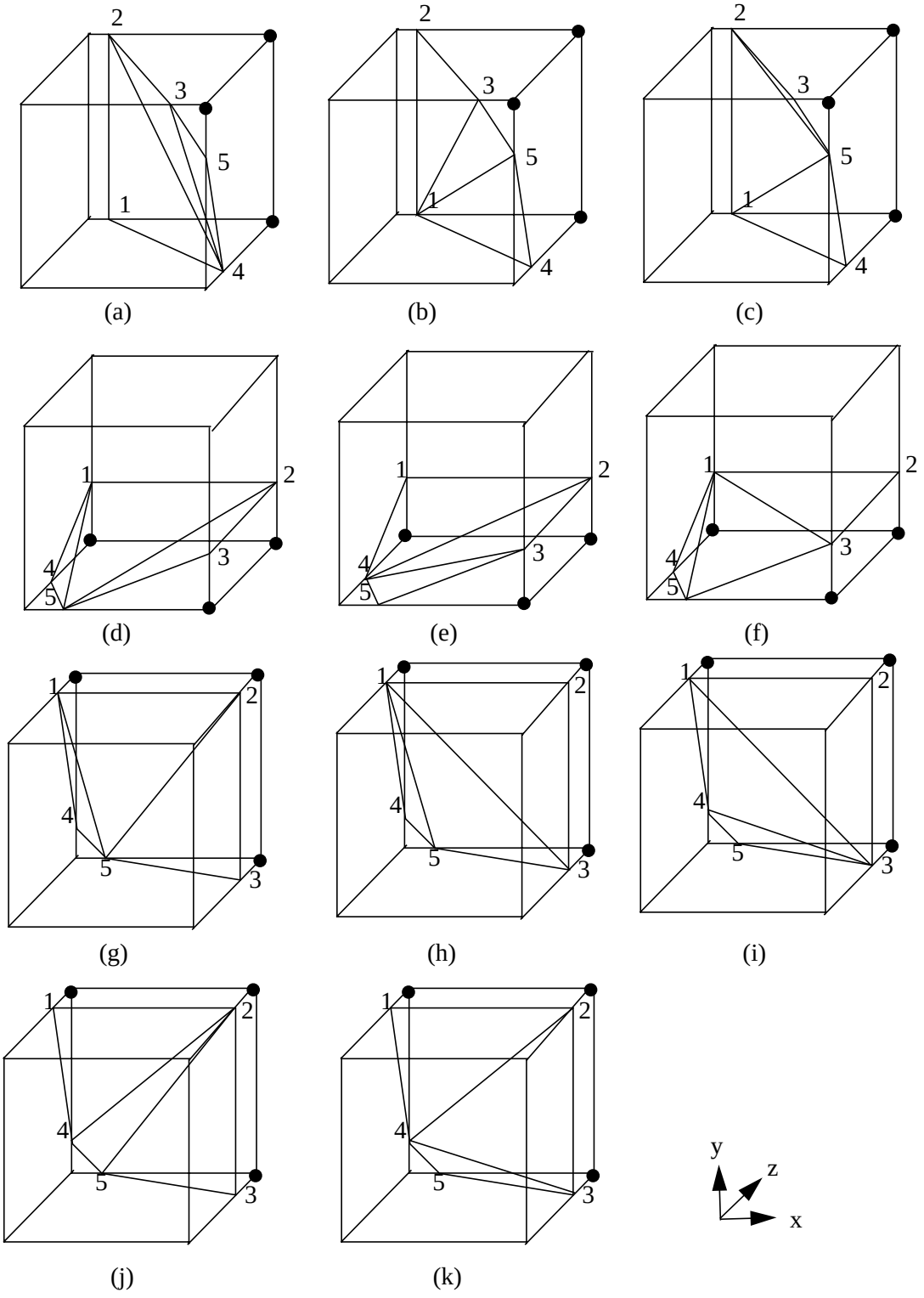

Figure 9. Representatives of Configuration 5

I/O Face		Representative	Sequence of Vertices
I	[X, -Y]	a	(45)S3S2S1
		a	(54)3S2S1
	[X,+Y]	b	(45)132
		b	(54)S132
	[-Y, X]	a	(14)2S3S5
		a	(41)S2S3S5
	[-Y,+Y]	c	(14)S523
		c	(41)523
	[+Y, X]	b	(23)154
		b	(32)S154
	[+Y,-Y]	c	(23)S514
		c	(32)514
II	[-X,+X]	d	(14)S523
		d	(41)523
	[-X, Y]	e	(14)2S3S5
		e	(41)S2S3S5
	[+X,-X]	d	(23)S514
		d	(32)514
	[+X, Y]	f	(23)154
		f	(32)S154
	[Y, -X]	e	(45)S3S2S1
		e	(54)3S2S1
	[Y,+X]	f	(45)132
		f	(54)S132
III	[-X,+X]	g	(14)S523
		g	(41)523
	[-X, -Y]	j	(14)253
		j	(41)S253
	[-X, +Y]	h	(14)S5S3S2
		h	(41)5S3S2
	[+X,-X]	g	(23)S514
		g	(32)514
	[+X, -Y]	i	(23)1S4S5
		i	(32)S1S4S5
	[+X,+Y]	j	(23)S5S4S1
		j	(32)5S4S1
	[-Y,- X]	g	(35)241
		g	(53)S241
	[-Y,+X]	i	(35)S4S1S2
		i	(53)4S1S2
	[-Y,+Y]	k	(35)S421
		k	(53)421
	[+Y,-X]	h	(12)S3S5S4
		h	(21)3S5S4
	[+Y,+X]	j	(12)4S5S3
		j	(21)S4S5S3
	[+Y,-Y]	k	(12)435
		k	(21)S435

Table 4. Sequences of Vertices for Configuration 5

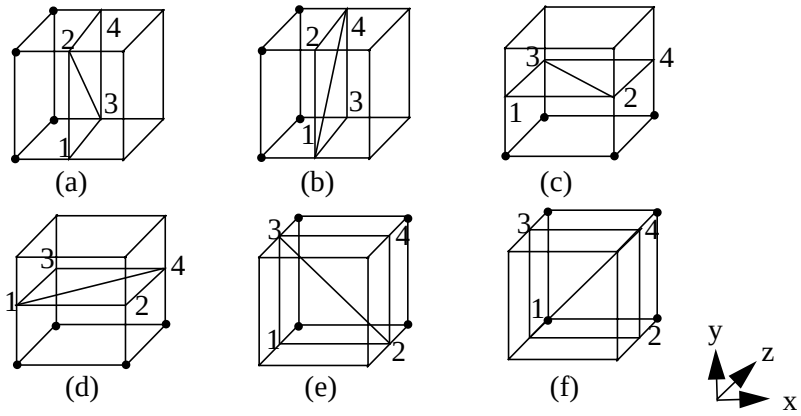

Figure 10. Representatives of Configuration 8

I/O Face		Representative	Sequence of Vertices
I	[-Y,+Y]	a	(13)24
		b	(31)42
	[+Y,-Y]	b	(24)13
		a	(42)31
II	[-X,+X]	c	(13)24
		d	(31)42
	[+X,-X]	d	(24)13
		c	(42)31
III	[-X,+X]	e	(13)24
		f	(31)42
	[-X, -Y]	f	(13)S4S2
		f	(31)4S2
	[-X,+Y]	e	(13)2S4
		e	(31)S2S4
	[+X,-X]	f	(24)13
		e	(42)31
	[+X,-Y]	e	(24)S3S1
		e	(42)3S1
	[+X,+Y]	f	(24)1S3
		f	(42)S1S3
	[-Y, -X]	f	(12)S4S3
		f	(21)4S3
	[-Y,+X]	e	(12)3S4
		e	(21)S3S4
	[-Y,+Y]	f	(21)43
		e	(12)34
	[+Y,-X]	e	(34)S2S1
		e	(43)2S1
	[+Y,+X]	f	(34)1S2
		f	(43)S1S2
	[+Y,-Y]	f	(34)12
		e	(43)21

Table 5. Sequences of Vertices for Configuration 8

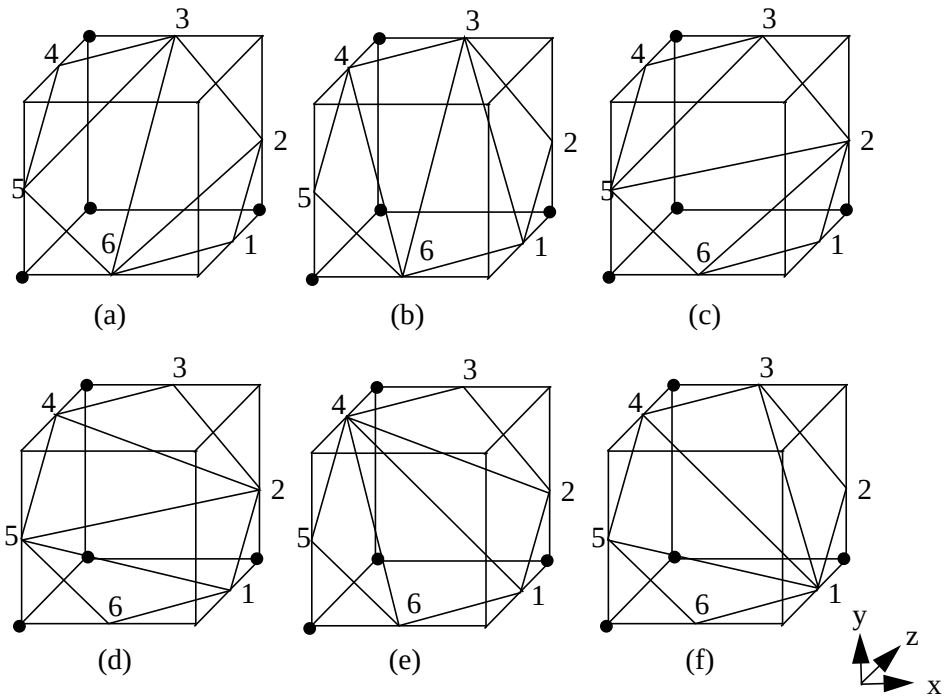

Figure 11. Representatives of Configuration 9

I/O Face	Representative	Sequence of Vertices
[-X,+X]	a	(45)3621
	b	(54)6312
[-X, -Y]	a	(45)362S1
	a	(54)S362S1
[-X,+Y]	e	(45)S6S1S2S3
	e	(54)6S1S2S3
[+X,-X]	a	(12)6345
	b	(21)3654
[+X,-Y]	f	(12)S3S4S5S6
	f	(21)3S4S5S6
[+X,+Y]	a	(12)635S4
	a	(21)S635S4
[-Y, -X]	c	(16)253S4
	c	(61)S253S4
[-Y,+X]	f	(16)S5S4S3S2
	f	(61)5S4S3S2
[-Y,+Y]	c	(16)2534
	d	(61)5243
[+Y,-X]	e	(34)2S1S6S5
	e	(43)S2S1S6S5
[+Y,+X]	c	(34)S526S1
	c	(43)526S1
[+Y, -Y]	d	(34)2516
	c	(43)5261

Table 6. Sequences of vertices for configuration 9

Data Set	No. of Tri.	Timing(s)	No. of Tri. / S	Avg L of Mesh	% of Swap / Tri.
CThead1	1,035,790	6.43	161K	48	9%
CThead2	837,944	5.66	148K	28	17%
CThead3	1,127,114	7.18	157K	43	12%

Table 7. Test Results from SMSR Algorithm Implementation