

A novel approach to extract triangle strips for iso-surfaces in volumes

Jagannathan Lakshminpathy
Advanced Visualization Lab,
Indiana University, USA.
jlakshmi@iupui.edu

Wieslaw L. Nowinski
Biomedical Imaging Lab,
Institute for Infocomm Research,
Singapore.
wieslaw@i2a-star.edu.sg

Eric A. Wernert
Advanced Visualization Lab,
Indiana University, USA.
ewernert@indiana.edu

Abstract

The Marching Cubes (MC) algorithm is a popular approach to extract iso-surfaces from volumetric data. This approach extracts triangles from the volume data for a specific iso-value using a table lookup approach. The lookup entry in the MC is a name-value pair, where the name is a number that uniquely identifies a cube topology and the value is the set of triangles for that topology. The MC applies a divide-and-conquer strategy by subdividing the volume into cubes with voxels at each corner of the cube and processes these cubes in a specific order. Thus, for a user specified iso-value, the MC looks up triangles for each cube and thereby generates the whole iso-surface. Most modern graphics hardware renders triangles faster if they are rendered collectively as triangle strips as opposed to individual triangles. Therefore, in this paper we have modified the MC lookup table approach such that the name is the cube topology and the value is a sub-surface piece(s) and its face-index representation. At the time of extraction we tessellate the sub-surface pieces by considering the pieces in the neighboring cubes using the face-index representation and then triangulate these tessellated sub-surface pieces into triangle strips. Our approach is superior to the existing approaches. Its features include: (1) simplicity, (2) procedural triangulation which avoids painful pre-computation, and (3) face-index representation of surface pieces that enables an efficient connection mechanism.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Geometric algorithms; I.3.7 [Three-Dimensional Graphics Realism]: Virtual Reality; I.3.m [Miscellaneous].

Keywords: Marching Cubes, triangle strips, sub-surfaces, iso-value, iso-surface.

1 Introduction

A 3-D display of a surface reconstructed from volume data involves surface generation and surface rendering. Surfaces with millions of polygons are quite common when reconstructed from 3-D data acquisition grids, such as MR or CT, or when using human body models, such as brain atlases [Nowinski 2001]. Even computers with very specialized and highly optimized geometric engines have problems in rendering them in realtime. Two major approaches are identified to speed up the rendering phase: a) merging a set of neighboring triangles which share at least two vertices into a triangle strip and thereby rendering them as a single strip of triangles, and b) lowering the number of vertices and reducing the complexity of connectivity. In this paper we are dealing with the former approach.

Rendering a stripped surface is more efficient than a non-stripped surface because the shared vertices of two neighboring triangles have to be sent only once as opposed to two times in a non-stripped surface. Hence, rendering of a stream of triangles in a stripped mode is up to three times faster than in a non-stripped mode. Rendering in a stripped mode is available in most industrial graphics standards.

Various attempts have been made to extract iso-surfaces from 3-D volumes. [Herman et al. 1983] created surfaces from cuberilles. [Meagher et al. 1982] used an octree representation to compress the storage of 3-D data, allowing rapid manipulation and display of 3-D data. [Farrel et al. 1983] and [Hohne et al. 1986] used ray casting to find the 3-D surface. The Marching Cubes (MC) by [Lorensen et al. 1987] creates a fine polygon represented surface by modifying [Wyvill et al. 1986]. The MC gives 15 cube patterns based on voxel values at the corners. Later some rigorous analysis [Wilhems et al. 1990][Natarajan et al. 1994][Durst et al. 1988] introduced 4 new cases to the basic 15 cases (cases 0 to 14) in the MC called cases 15, 16, 17, and 18. All of these algorithms exploit the complementary and rotational symmetry to reduce the number of cases to be considered.

Later, [Akeley et al.] have post processed a triangulated surface into a stripped surface. [Deering et al. 1995] have proposed the use of generalized triangle strips for compressing connectivity information in geometric polygonal models. [Arkin et al.] have proved that testing, if a triangulation is Hamiltonian, is NP-complete. [Evans et al. 1996] have proved that the problem of finding a sequential triangulation is NP-complete. [Bar-Yehuda et al. 1996], [Evans et al. 1996] present the extent to which they can increase the stack size to reduce the duplication of vertices during rendering. All the above methods can be classified either as: (1) iso-surface extraction method, or (2) post process the extracted iso-surface triangles to triangle-strips. Post-processing an already extracted iso-surface has many pitfalls. For instance, if we are to identify a shared edge between a pair of polygons, one with m edges and the other with n edges, if there exists one then it requires $O(m*n)$ time. On the other hand, the methods that output triangle strips dynamically can make use of the lattice representation to identify the common edge in $O(m+n)$ time (see the discussion section 4.1 for more details).

[Zhou et al. 1995] have suggested a method of generating triangle strips dynamically based on the occurrence of cube topology (cube pattern). Even this method is not without pitfalls. They argue that only 5 specific configurations (configurations 1, 2, 5, 8, 9) of the 15 basic MC cube configurations occur frequently and hence they process only those configurations to produce triangle strips. Even when processing those specific configurations, they have to precompute the triangulation for every possible combinations of a leading and a trailing edge for each surface. For Configuration 1, they precomputed the triangulation for 4 possible cases. Similarly for Configuration 2, 5, 8 and 9 they have to precompute 28, 48, 32 and 24 possible triangulations. This method has many disadvantages:

- (1) Only pre-selected surface pieces are processed for triangle strips.

- (2) Numerous triangulation cases have to be considered for each preselected surface piece. Nevertheless rotational and complement symmetries might increase those triangulation cases further.
- (3) The occurrence of an unselected surface piece in middle of a strip might limit the progress of the strip.

In this paper, we present a simple yet effective approach that generates triangle-strips for a specified iso-surface. We first create a lookup table where each entry refers to a cube topology, similar to the MC, but each entry stores the face-index representation for each surface piece inside the cube (see section 2 for detail). Our approach has two phases: (1) Lookup and tessellation phase and (2) Triangle strip generation phase. In phase 1, we consider a cube at a time and lookup for the face-index representations for each surface piece in the cube. Then, we create an instance of the surface piece, and connect it to its neighbor (if we can). In phase 2, triangle strip generation phase, we pick each strand of connected surface pieces and generate triangle strips procedurally. This paper is organized as follows. In section 2, we describe our method in detail. In section 3, we present the results followed by discussion and conclusion.

2 Our Approach

We first present the iso-surface extraction problem and a typical approach to it followed by our approach. Initially we explain our approach in 2-D for simplicity with simple data structures and later we extended the problem into 3-D. In sub-sections 2.1 and 2.2 we discuss 1 and 2 phases in detail.

Iso-surface extraction deals with the problem of generating a surface from a scalar function. We are given a 3-D volume of scalar data samples (l, m, n) with $0 < l < L$, $0 < m < M$, $0 < n < N$ and an "isovalue" k . Volume data is decomposed into cubes with data samples at the corners of the cubes. The Volume space of each cube is partitioned by polygons based on the sample values at corners into region(s) inside and region(s) outside of the surface that is reconstructed. The polygons that interface the binary regions form the reconstructed surface. The edges of the polygons can be either contained inside the cube or on the face of the cube. The edges that lie on the faces of the cube define a contour in 3-D space that encapsulates a connected set of polygons in the cube. We reserve the word *sub-surface piece* for the connected set of polygons and word *sub-surface boundary* for the contour.

The MC algorithm [Lorenson et al. 1990] processes the cubes in a specific order. Since there are eight vertices in each cube and two states, inside and outside, sub-surfaces can intersect a cube in 256 different ways. Two different symmetries namely the rotation and the complement of the cube enable us to reduce the problem to 15 cases. The MC generates polygons of the reconstructed surface by a table lookup approach where each entry in the table corresponds to a topology of the cube. Though the triangle set that comes from neighboring cubes can be connected, the lack of face index representation in those sub-surface pieces makes the connection impossible or expensive.

First let us consider the problem in 2-dimensions. Let the 2-D grid define the samples for the scalar function $f(x, y)$ at each grid point. Let us assume that we are extracting the iso-curve $f(x, y) = k$. Let the grid points inside the contour refer to sample values less than k and similarly the points outside the contour refer to sample values greater than k . The 2-D curve in Figure 1 refers to a piecewise approximation of the iso-curve. Our 2-D algorithm is expected to extract the curve as connected line segments. Assuming that we are using the row dominant marching order in

figure 1 the bottom-left square will be visited first and the top-right square last. In our 2-D case, we are dealing with squares as opposed to a hexahedron in a 3-D case. Each square has 4 corners and each corner can be in one of two states, inside or outside, giving 16 possible square topologies to consider. We list 3 types of line segments as in the top row of figure 1(b) $\{T_0, T_1$ and $T_2\}$. We claim it is possible to represent any 2-D iso-contours extracted from a 2-D grid with those 3 types. The bottom row in fig 1(b) refers to 2 of the 16 cases that are of no interest to us, as they generate zero line segments.

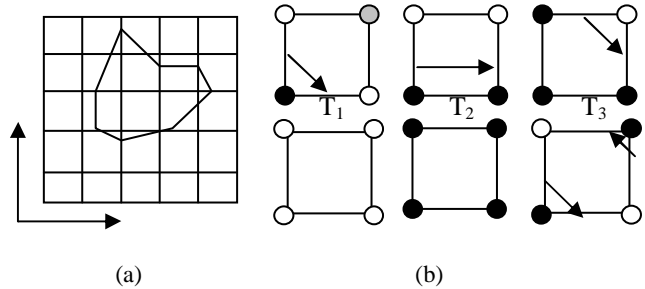


Figure 1. **A sample 2-D grid and 3 distinct line segment types (T_0, T_1 and T_2).** (a) Piecewise approximation of an iso-contour $f(x, y) = k$. Two arrows on the left show the directions of the lattice coordinates. The bottom-left cell is referred as C_{11} , the bottom-right cell is C_{15} , the top-left cell is C_{51} and the top-right cell as C_{55} (considering the row-column marching order). (b) The top row shows all possible line segment types T_0, T_1 , and T_2 (left to right) and the bottom row shows 3 of 16 possible square topologies. The dark corners refer to samples with value $< k$ and white corners to samples with value $> k$. Gray corners indicate that the sample state, which can be either dark or white, doesn't affect the type of the line segment. First two cases of the bottom row generate no line segments and the third case can be represented as two separate instance of type T_0 (see section 2 for more details).

Let us take the first case from the right in the bottom row of figure 1(b). This particular case has 2 line segments. For instance, if we name the bottom, right, top and left edges of the square as 0, 1, 2 and 3 respectively, then the directed line segments for the above case can be conveniently referred by two ordered pairs $\{3, 0\}$ and $\{1, 2\}$. Now let us consider the 90° anti-clockwise rotational instance for the above case. Then the corresponding face-index representations become $\{0, 1\}$ and $\{2, 3\}$. Likewise, an 180° anti-clockwise rotation corresponds to a $\{1, 2\}$ and $\{3, 0\}$. Thus, a face-index representation (i, j) and a face-index representation for its corresponding rotational instance is a simple modulo-4 shift operation, $\{(i+k) \bmod 4, (j+k) \bmod 4\}$ where $k = \text{angle}/90$. Let us refer to this rotational operation by $\text{SHIFT } k$. For example, the first case from the right in the bottom row of figure 1(b) shows two line segments that are instances of T_0 namely T_0 and $T_0 \text{SHIFT } 2$.

Before we explain the details of the connection process, we define a bucket data structure that is required to collect the connected line segments. A bucket is a sequence of vertices where the first vertex in the sequence is referred to as the *leading vertex* and the last vertex in the sequence as the *trailing vertex*. The leading and trailing vertices have references to the square they originated from, e.g. C_{34} , and to the face-index representation, e.g. $T_2 \text{SHIFT } 1$.

For the above 2-dimensional grid example, in row 2 square C_{22} will generate an edge $T_0 \text{SHIFT } 2$ and square C_{23} will generate a

segment T_1 SHIFT 2. From their corresponding face-index representations $\{1, 2\}$ and $\{1, 3\}$, respectively, and the fact that C_{22} and C_{23} are left-right neighbors we know they share a common vertex V_1 at edge 1 of square C_{22} and edge 3 of C_{23} square and, therefore, the two edges can be connected and our bucket data structure would look as $\{V_0, V_1, \text{ and } V_2\}$ with the leading V_0 from $\{C_{23}, T_1 \text{ SHIFT } 2\}$ and trailing vertex V_2 from $\{C_{22}, T_0 \text{ SHIFT } 2\}$. Likewise, in row 2, square C_{24} will generate a segment T_0 SHIFT 3, with a face-index representation $\{2, 3\}$. This segment will share a leading vertex V_0 and hence the bucket can be updated as $\{V_4, V_0, V_1, \text{ and } V_2\}$ with the leading vertex V_4 from $\{C_{24}, T_0 \text{ SHIFT } 3\}$ and trailing vertex V_2 from $\{C_{22}, T_0 \text{ SHIFT } 2\}$. Similarly, in row 3, square C_{32} will generate a segment T_1 SHIFT 1, with a face-index representation $\{0, 2\}$. This segment will share the trailing vertex V_2 and hence the bucket can be updated as $\{V_4, V_0, V_1, V_2, \text{ and } V_3\}$ with the leading V_4 from $\{C_{24}, T_0 \text{ SHIFT } 2\}$ and trailing vertex V_3 from $\{C_{32}, T_1 \text{ SHIFT } 1\}$. Likewise, in row 3, square C_{34} will generate a segment T_2 SHIFT 3, with a face-index representation $\{1, 0\}$. This segment will share the leading V_4 and therefore the bucket can be updated as $\{V_5, V_4, V_0, V_1, V_2, \text{ and } V_3\}$ with the leading V_5 from $\{C_{34}, T_2 \text{ SHIFT } 3\}$ and trailing vertex V_3 from $\{C_{32}, T_1 \text{ SHIFT } 1\}$. This process will continue until we finish processing the square C_{55} . At that point the sequences of vertices in bucket will represent the extracted iso-contour.

For the 3-D case the approach is very similar. We will deal with hexahedrons in the 3-D case as opposed to squares in the 2-D case. Similarly we deal with sub-surface piece(s) inside the cube in the 3-D case as opposed to line segment(s) inside the square in the 2-D case. Each hexahedron has 8 corners and as before each corner can be in one of two states, inside or outside, resulting in 256 possible cube topologies to consider. Figure 2 lists all possible sub-surface topology geometries within a cube. Let us name them as $\{T_0, T_1, T_2, \dots, T_{11}\}$ in that order (left to right and top to bottom). As there is a single-degree of freedom in generating rotational instances (clockwise or anti-clockwise) in the 2-D case, there are 3-degrees of freedom (rotation about X, Y and Z) in generating rotational instances in the 3-D case. So the rotational instance of any of the listed sub-surface can be represented by a SHIFTX, SHIFTY, SHIFTZ operators (note that SHIFTX, SHIFTY and SHIFTZ operators are not a simple modulo 4 operation). Figure 3 below shows two such examples.

2.1 Sub-surface Tessellation

The approach we just outlined is a two-phase approach: (1) lookup and tessellation phase and (2) triangle strip generation phase. In this sub-section we discuss the details related to the lookup and tessellation phase.

Like MC, we partition the volume into cubes and process the cubes one at a time in a specific order. We follow the row-column-slice dominant order. Like MC, our approach uses a lookup table (name-value pair). The name being the cube topology (a number between 0-255 that identifies the 256 possible cases) and the value is the face-index representation of each sub-surface piece inside the cube. The face-index representation of the sub-surface piece is an ordered sequence of indices that refer to the edges of the cube that contain the tie-points but not the actual tie-points. We introduce three data structures: *PieceHeap*, *Bucket* and *NeighborTable*. *PieceHeap* structure stores every generated surface piece and associates a unique index to it. *NeighborTable* will have as many as $row * column * 2$ entries, one each for a cube in the previous slice and in the current slice. Each entry in the *NeighborTable* stores piece identifiers, an index that refers to a piece in the *PieceHeap* structure, for every piece in the cube.

For each cube, with voxel values and gradients, we lookup the face-index representation of each surface piece contained in the cube. We then compute the exact geometric coordinates and geometric normal for each tie-point in the face-index representations. We insert the above vertices and normals in a hash-table, and we then insert their corresponding indices in the *vertex_id* field of the *SubsurfacePiece* data structure. As shown in step 3 (see the algorithm below), the computed *SubSurfacePiece* is inserted into the *PieceHeap* and the corresponding index is stored in the *NeighborTable* entry for the cube and also in a newly created *Bucket* (we assign a new bucket for each newly created piece). The *Bucket* data structure contains the connected sequence of pieces (also referred to as a *strand*).

A step-by-step procedure, *Connect*, that attempts to connect a sub-surface piece S to its neighbors is shown below. First we search for the row neighbor. To accomplish this we look for pieces in the *NeighborTable* entry for the neighboring cube. For each piece $S_{i,j}$ in the neighboring cube, we actively search for connections as mentioned in steps 10 through 20. A piece in a bucket is considered a leading piece if it is the first in the bucket. Similarly a trailing piece is the last piece in the bucket. At step 8, we look for pieces in the neighboring cube. We repeat the steps 8 through 20 for other neighbors namely the column and the slice neighbors. In our *SubsurfacePiece* data structure if the *leading_edge* field is set, to a valid index, then that piece ceases to be a leading piece. Likewise if the *trailing_edge* field is set then the piece ceases to be a trailing piece. In step 12, *leading_edge* and *trailing_edge* fields are used to check if the piece is a leading or trailing piece. These fields are updated after every connection.

```

PieceHeap
{
    int maxNumber;
    int n_pieces;
    SubsurfacePiece* pieces;
};

Bucket
{
    int maxNumber;
    int n_pieces;
    int* pieces;
};

NeighborTable
{
    int n_pieces;
    int* pieces;
};

SubSurfacePiece
{
    char piece_type;
    int* vertex_id;
    int leading_edge;
    int trailing_edge;
};

1. For each Cube  $C_{i,j,k}$  in the volume dataset
2.   For each sub-surface  $S_k$  in  $C_{i,j,k}$ 
3.     Insert  $S_k$  into the Heap and get the id;
4.     Insert the id into the NeighborTable
4.     Create a new Bucket B;
4.     Insert the id of  $S_i$  in to B;
5.     Connect( $S_k, B, C_{i,j,k}$ );
6. For each bucket  $B_i$ 
7.   GenerateTstrips( $B_i$ );
8.   Output the vertices and triangle-strips;

7. PROCEDURE Connect( SubSurfacePiece S, Bucket B, Cube c)

//Look for connections in the already visited neighbors
//of C. We assume the row-column-slice dominant traversal.
//We refer the cube  $C_{i-1,j,k}$  as the row neighbor of  $C_{i,j,k}$ . Simi-
//larly  $C_{i,j-1,k}$  and  $C_{i,j,k-1}$  as column and slice
//neighbors of  $C_{i,j,k}$  resp.
8. For the row neighbor  $C_k$  of C

//There can be a maximum of only 4 sub-surfaces in  $C_k$ 
//and of those 4 only 2 sub-surfaces can touch C if
//at all.
9. For each sub-surface  $S_{i,j}$  inside  $C_k$  that touch C

```

```

10.   if  $S_{i,j}$  is fully connected then continue;
11.   if ( $S_{i,j}$  is a trailing piece in a Bucket AND
12.      $S$  is a leading piece of B) then
13.     Merge the two Buckets into one;
12.   Update  $S_{i,j}$ .trailing_edge,  $S$ .leading_edge;
12.   Continue;

13.   if ( $S_{i,j}$  is a leading piece in a Bucket AND
14.      $S$  is a trailing piece of B) then
14.     Merge the two Buckets into one;
14.     Update  $S_{i,j}$ .trailing_edge,  $S$ .leading_edge;
14.     Continue;

15.   if ( $S_{i,j}$  is a trailing piece in a Bucket AND
16.      $S$  is a trailing piece of B) then
16.     Reverse the Bucket that contains the piece  $S_{i,j}$ 
17.     Merge the two Buckets into one;
17.     Update  $S_{i,j}$ .trailing_edge,  $S$ .leading_edge;
17.     Continue;

18.   if ( $S_{i,j}$  is a leading piece in a Bucket AND
19.      $S$  is a leading piece of B) then
19.     Reverse the Bucket that contains the piece  $S_{i,j}$ 
20.     Merge the two Buckets into one;
20.     Update  $S_{i,j}$ .trailing_edge,  $S$ .leading_edge;
20.     Continue;

21.   Repeat steps 9 thru. 20 for each column and slice
neighbors of C
.....
.....
.....
100.   END PROCEDURE

```

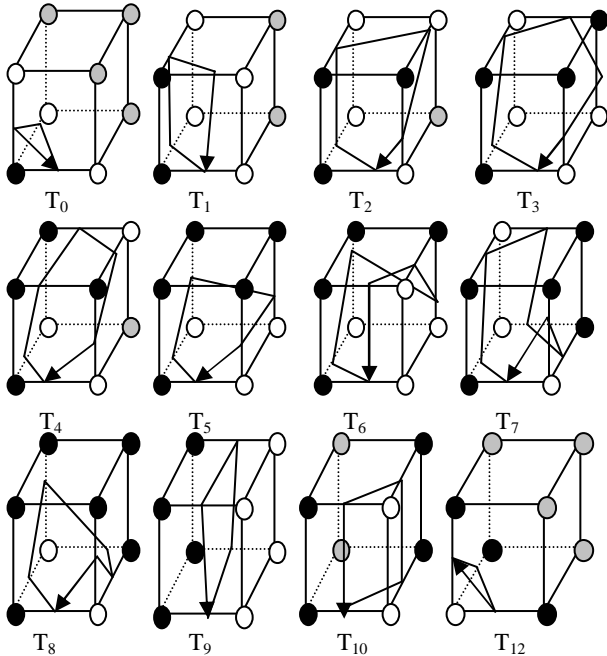


Figure 2. All possible sub-surface topology geometries. Dark corners refer to the samples inside a 3-D contour and white corners to the samples outside the 3-D contour. The gray corners (don't care state) are of no particular interest (as it won't affect that particular sub-surface shape) as we are interested only in listing all possible surface topology geometries, not all cube topologies.

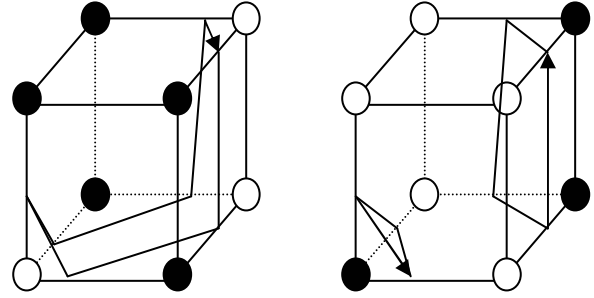


Figure 3 Two of the 256 cube topologies: (a) Sub-surface piece inside this cube is a rotational instance of T_7 and its face-index representation is given by T_7 SHIFTZ 3 (b) This cube has two sub-surface pieces and their corresponding face-index representations are T_0 and T_1 SHIFTZ 2.

2.2 Triangle-strip Generation

The triangle-strip generation phase begins at step 6. In this phase for each *bucket* or strand, created in the tessellation phase, we compute the triangle strips procedurally and output those triangle-strips into a file. Typical 3-D graphics libraries, e.g. OpenGL, draw connected group of triangles or triangle-strips [Woo et al. 1993] for a sequence of input vertices as follows. If $\{V_1, V_2, \dots, V_n, V_{n+1}, V_{n+2}, \dots, V_m\}$ is a sequence of vertices fed to graphics pipeline, then for odd n , the vertices V_n, V_{n+1}, V_{n+2} define the n^{th} triangle. For even n , the vertices V_{n+1}, V_n, V_{n+2} define the n^{th} triangle. A total of $m-2$ triangles are drawn for the sequence. Figure 4a shows a typical sample piece in a bucket. Figure 4 b and c show two possible triangulations for the piece. In figure 4b segment V_6V_5 is the leading edge and V_3V_4 is the trailing edge and the vertex sequence for the triangulation shown in figure 4 (b) is $\{V_5, V_6, V_1, V_4, V_2$ and $V_3\}$. Similarly, in figure 4c segment V_6V_5 is the leading edge and V_4V_5 is the trailing edge and its corresponding vertex sequence for the triangulation is $\{V_6, V_5, V_1, V_5, V_2, V_5, V_3, V_5, V_4\}$. Note the repetition of the vertex V_5 in the sequence and the edge fanning effect around the vertex V_5 . We present the pseudo-code for a generic triangulation procedure for any arbitrary sub-surface piece with a leading and a trailing edge:

```

PROCEDURE Triangulate(SubSurfacePiece piece,
VertexSequence vertices)
N = GetNumberOfVertices(piece.piece_type);
Start[0] = piece.leading_edge;
Start[1] = (Start[0]+1)%N;
End[0] = piece.trailing_edge;
End[1] = (End[0]+1)%N;
while((Start[1] != End[0]) || (Start[0] != End[1]))
if(Start[0] != End[1])
Start[0] = ((Start[0]-1)<1)?(N):(Start[0]-1)
Vertices.Add(piece.vertices[Start[0]]);
else
Vertices.Add(piece.vertices[Start[0]]);
if(Start[1] != End[0])
Start[1] = ((Start[1]+1)%N==0)?1:(Start[1]+1)
Vertices.Add(piece.vertices[Start[1]]);
else
Vertices.Add(piece.vertices[Start[1]]);

```

Function `GetNumberOfVertices` gets the number of vertices for a given piece type. Initially, the `Start` and `End` arrays are initialized to the leading and trailing edge for the piece under consideration (Vertex and edge indices can be easily converted from one to another using the modulo). We update each vertex of the start edge as we pass the loop each time until either the `Start[0]` becomes `End[1]` or `Start[1]` becomes `End[0]`. If `Start[0]` becomes `End[1]` before `Start[1]` becomes `End[0]` then we fan around `Start[0]` until `Start[1]` becomes `End[0]`. Similarly if

$Start[1]$ becomes $End[0]$ before $Start[0]$ becomes $End[1]$ then we fan around $Start[1]$. This will go on until $Start[1] = End[0]$ and $Start[0] = End[1]$. We repeat the above procedure for each piece in the *bucket*, Refer to steps 6,7 and 8 in the listing in Section 2.1. Note that the leading or the trailing edges for the end pieces in the sequence will be uninitialized .In that case we arbitrarily choose the trailing or leading edge.

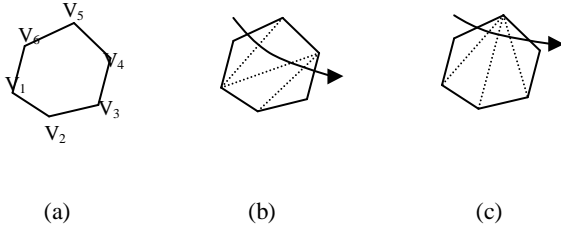


Figure 4. **A Sample sub-surface piece and its two possible triangulations:** (a) A sample piece (b) Best possible triangulation for the piece and (c) Another triangulation for the piece with duplicate vertices.

3 Results

The proposed algorithm has been implemented and tested with several datasets. The results for two types of data, analytical object and scanned data (CT), are given in Figure 5. We presented two images for each data set: shaded and color mapped. The purpose of the color-mapped images is to show the progress of the triangle strips. We picked distinct color values, as much as possible, for neighboring triangle strips. In the analytical data category we tried the knot data that is shown below. We tried the data for 256x256x256 resolution. In the scanned data category, we tried several data sets, but we presented the timing results for the famous CT Skull and CT Head data sets only. We ran our application on a remote high-end graphics workstation namely SGI Onyx multi-node computer (though it doesn't have to be a high-end workstation) while the final rendered image was displayed on my local desktop PC. The remote application reads the pre-computed triangle strips from a file and issues GL (required to render the triangle strips) calls across the network through a GLX protocol. The GLX client running on the local desktop receives the GL calls and renders them with the help of local graphics hardware. Table 1 summarizes the timing results. We observed a performance increment factor between 2.0 and 2.8.

#	Dataset	Resolution	# of triangles	# of strips	Sec./fr.	
1	Knot	256x256x256	1204001	57099	7	14
2	CT Skull	256x256x128	800025	32808	3.5	9.8
3	CT Head	256x256x128	1124156	46946	5	12

Table 1

4 Discussion

We have outlined a new method that extracts the iso-surface for a user specified iso-value. This approach uses a MC lookup table approach, but unlike the MC, the lookup table entries are modified to store face-index representation of sub-surface pieces that are

contained in the cube. Another advantage of the method lies in the triangulation phase.

4.1 Face-index Representation:

Let us name the edges of the cube as shown in Fig 6. Moreover, let us name the six faces of the cube (1,2,3,4), (9,10,11,12), (1,8,9,5), (6,3,7,11), (5,2,6,10) and (8,4,7,12) as $-Z$, $+Z$, $-X$, $+X$, $-Y$ and $+Y$, respectively. Face-index representation for T_0 shown in fig 2 is {1,5,2}. Similarly for T_0 SHIFTZ 2 is {8, 4, 3}. From a face-index representation for any sub-surface piece with n edges we can identify an edge (or edges) on a particular face of the cube in $O(n)$ time. At step 9, we look for a piece in the neighboring cube (row or column or slice neighbor) that shares an edge with the current piece under consideration.

From the face-index representation (sequences of indices that refer to the edges of the cube that contain the tie-points) of the pieces we can identify the common edge in $O(m+n)$ time if the number of edges in the neighboring piece and the current piece is m, n respectively. In post processing methods, a similar operation takes $O(m*n)$ time.

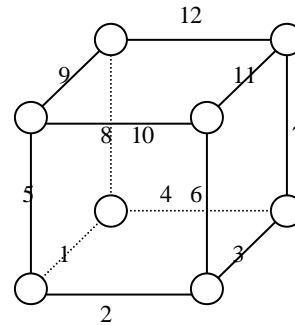


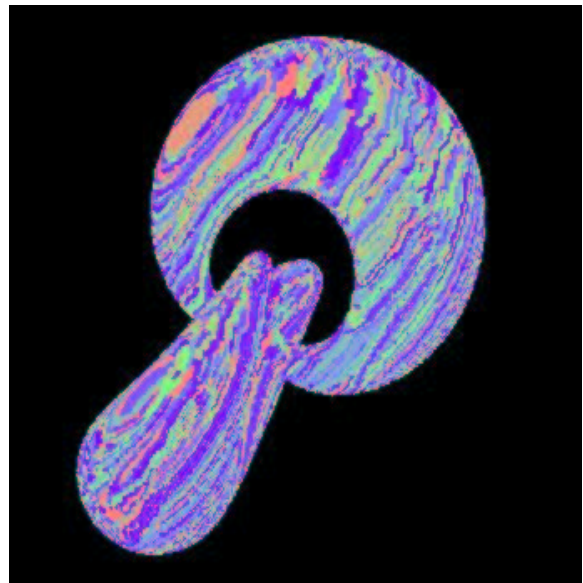
Figure 6. **Face index Representation**

4.2 Procedural Triangulation:

When triangulating sub-surface pieces two approaches are possible: (1) Pre-compute triangulation and (2) Compute triangulation procedurally. Pre-computing triangulation for every possible sub-surface piece and all possible combination of leading and trailing edge is tedious and error prone. Our method employs procedural triangulation and we have listed the pseudo-code for procedural triangulation in section 2.2. In phase 1, we connect the sub-surface pieces that share an edge into a sequence and in phase 2, we pick each strand of sub-surface pieces and triangulate each piece procedurally picking one piece at a time. We could have designed our algorithm as single phase as opposed to our two-phase approach but it was just a matter of preference.



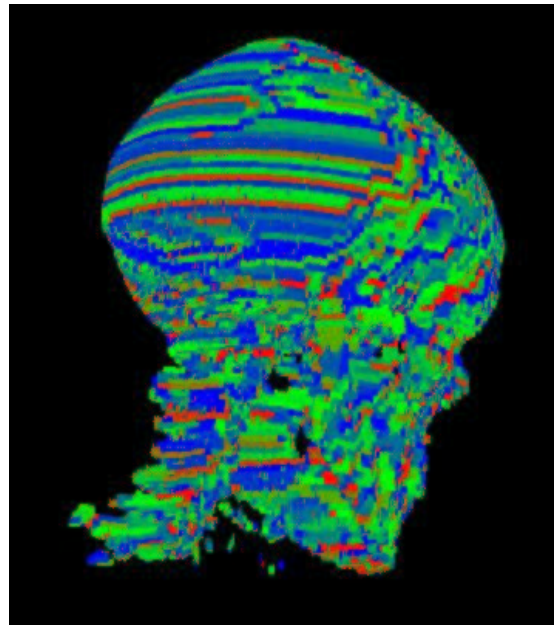
5 (a)



5(b)



5(c)



5(d)

Figure 5. **Rendered Images:** (a) & (c) Gouraud shaded rendering of knot and skull data respectively (b) & (d) Color coded rendering of knot and skull data (no two neighboring strips were mapped with the same color for clarity) respectively.

5 Conclusion

In this paper we have presented a new approach that extracts triangle strips for iso-surface in the 3-D data. This approach uses a MC lookup table approach, but, unlike the MC, the lookup table entries are modified to store face index representation of sub-surface pieces that are contained in the cube. We presented a two-phase method. In phase 1, we subdivide the volume into cubes and we look up the face-index representation of each piece inside the cube and connect those pieces into sequences or strands. In phase 2, we process each strand sequentially by employing procedural triangulation by picking a piece from the strand at a time.

Acknowledgements

This research was supported in part by the Indiana Genomics Initiative (INGEN). The Indiana Genomics Initiative (INGEN) of Indiana University is supported in part by Lilly Endowment Inc. The Institute of Systems Science, National University of Singapore, funded the initial part of this research. Michael J. Boyles and Dr. Shiao-fen Fang helped review the paper. The authors are indebted to those who contributed to this paper either directly or indirectly.

References

- G. Wyvill, C. McPheeters and B. Wyvill: *Data Structure for Soft Objects*, The Visual Computer, 2 (1986), pp. 227-234.
- C. Zhou, R. Shu and M.S. Kankanhalli: *Selectively Meshed Surface Reconstruction*. Computers & Graphics, 19, 6 (1995), pp. 793-804.
- M. Woo, T. Davis and J. Neider: *OpenGL Programming Guide*. Addison-Wesley Publishing Company, Reading, MA, 1993.
- W. E. Lorensen and H.E. Cline: *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*, Computer Graphics 21, 4, (July 1987), pp. 38-44.
- J. Wilhelms and A. Van Gelder: *Topological Considerations in Isosurface Generation Extended Abstract*, Computer Graphics 24, 5, (Nov. 1990), pp. 79-86.
- B.K. Natarajan: *On generating topologically consistent isosurfaces from uniform samples*. Visual Computer 11, 1994, pp. 52-62.
- P. Ning and J. Bloomenthal: *An Evaluation of Implicit Surface Tilers*. IEEE Computer Graphics and Applications. (Nov. 1993), pp. 33-41.
- K. Akeley, P. Haeberli, and D. Burns: *tomesh.c*: C Program on SGI Developer's Toolbox.
- E. Arkin, M. Hled, J.Mitchell, and S. Skiena. *Hamiltonian triangulations for fast rendering*. In Second Annual European Symposium on Algorithms, Springer-Verlag Lecture Notes in Computer Science, Vol. 855, pp 36-47.
- M. Deering: *Geometry compression*. Computer Graphics Proceedings, Annual Conference Series, (1995), pp 13-20.
- M.J. Durst: *Letters: additional reference to marching cubes*. Computer Graphics 22, 1988, pp 72-73.
- R. Bar-Yehuda and C. Gotsman. *Time/space tradeoffs for polygon mesh rendering*. ACM Transactions on Graphics, (1996), pp. 141-152.
- F. Evans, S. Skiena and A. Varshney: *Optimizing Triangle Strips for Fast Rendering*. Proceedings of IEEE Visualization, (1996), pp. 319-326.
- F. Evans, S. Skiena, and A. Varsheny. *Completing sequential triangulations is hard*. Technical report, Department of Computer Science, State Univ. of New York at Stony Brook, NY 11794-4400, 1996.
- G.T. Herman and J.K. Udupa: *Display of 3D Digital Images: Computational Foundations and Medical Applications*. IEEE Computer Graphics & Applications 3, 5 (Aug. 1983), pp. 39-46.
- E.J. Farrel: *Color Display and Interactive Interpretation of Three-Dimensional Data*. IBM J. Res. Develop 27, 4 (July 1983), pp. 356-366.
- D.J. Meagher: *Geometric Modeling Using Octree Encoding*. Computer Graphics and Image Processing 19, 2 (June 1982), pp. 129-147.
- W.L. Nowinski: *Computerized brain atlases for surgery of movement disorders*. Seminars in Neurosurgery 2001;12(2):183-194.
- K.H. Hohne and R. Bernstein: *Shading 3D-Images from CT Using Gray-Level gradients*. IEEE Trans. On Medical Imaging MI-5, 1 (March 1986), pp. 45-47