Piecewise Linear Hypersurfaces using the Marching Cubes Algorithm

Jonathan C. Roberts[a] and Steve Hill[b]

[a]University of Kent at Canterbury, Computing Laboratory, Canterbury, England, UK.

[b]Radan Computational Ltd. Ensleigh House, Granville Road, Lansdown, Bath, England, UK

ABSTRACT

Surface visualization is very important within scientific visualization. The surfaces depict a value of equal density (an isosurface) or display the surrounds of specified objects within the data. Likewise, in two dimensions contour plots may be used to display the information. Thus similarly, in four dimensions hypersurfaces may be formed around hyperobjects.

These surfaces (or contours) are often formed from a set of connected triangles (or lines). These piecewise segments represent the simplest non-degenerate object of that dimension and are named simplices. In four dimensions a simplex is represented by a tetrahedron, which is also known as a 3-simplex. Thus, a continuous n dimensional surface may be represented by a lattice of connected n-1 dimensional simplices.

This lattice of connected simplices may be calculated over a set of adjacent n dimensional cubes, via for example the Marching Cubes Algorithm. We propose that the methods of this local-cell tiling method may be usefully-applied to four dimensions and potentially to N-dimensions. Thus, we organise the large number of traversal cases and major cases; introduce the notion of a sub-case (that enables the large number of cases to be further reduced); and describe three methods for implementing the Marching Cubes lookup table in four-dimensions.

Keywords: Marching Cubes, four dimensions, hypersurfaces, surfaces

1. INTRODUCTION AND MOTIVATION

We live within three dimensional space; seeing our world via a two dimensional projection, which is reconstructed by our brain into a three dimensional model using motion, edge and depth cues. However, higher dimensions have been proposed and considered for many years; with the fourth dimension representing time, distance, a fourth spatial coordinate and even a spiritual realm.

Objects within four and higher dimensions can be generated from natural extensions to plane or solid geometry, with each three dimensional object (plane, cube, cone) having a four and higher dimensional equivalent (hyperplane, hypercube, hypercone). Data sets with higher dimensions can be generated from simulations, collated from statistics or sampled from real-life phenomena; many diverse fields-of-study provide data with copious variables that can be displayed in a number of dimensions using various imaging techniques.

Modern computer graphics provide the ability to view, interrogate and understand objects and phenomena that exist in higher dimensions. For example, an image of a Klein bottle, with a twisted surface, intersects itself within three dimensions, whereas within four dimensional space the bottle can be depicted without the self intersection.[1]

Geometry in higher dimensions can be (1) projected down to *lower* dimensions, using a variety of projection methods including parallel, perspective and central or (2) represented in other coordinate systems, including Parallel Coordinates[2] that depict the relationships and dependencies between N-Dimensional data (especially geometry) within a two dimensional parallel axis coordinate system.

Visualizations of n-dimensional data can be obtained by rendering the 'surface' of the data. The surface created is one dimension less than the original data: for example, the surfaces from two dimensional data create contour plots (one dimensional line segments in two dimensions) and three dimensional (volume) data produces two dimensional faces in three dimensions. Hence, from a four dimensional data volume a *hypersurface* is formed.

[a] j.c.roberts@ukc.ac.uk http://www.cs.ukc.ac.uk/people/staff/jcr/ [b] steve.hill@uk.radan.com

A two dimensional contour on a map, representing a particular height above sea-level, can be created using a continuous connection of straight line segments. Similarly, a continuous surface within three dimensions can be represented by a lattice of two-dimensional polygons. Therefore, a continuous hypersurface can be represented by a lattice of n-dimensional simplices. These simplex elements can be calculated from how the 'surface' intersects a set of adjacent n-dimensional cubes. A surface at a particular value (isosurface) through sampled data can be realised at the point of zero value, interpolated between any edge of an opposing sign. The signs at the n-cube vertices are found by thresholding the spatial data at a discrete data point. Consequently, hypersurfaces within higher dimensions can be depicted using a lattice of three-dimensional simplices (volumes), generated by local evaluation through a sample set of points.

This paper discusses the problems, requirements and some solutions in implementing an n-dimensional isosurface algorithm from spatial data, using local cell tiling methods. We focus on the generation of the n-dimensional geometry rather than the rendering or realistic-representation (using say higher-dimensional light) of the n-dimensional image.

Initially we present some background information. We then describe the algorithms and techniques: firstly from a theoretical viewpoint and secondly within a practical framework; we describe three table methods, extending the Marching Cubes Algorithm to four dimensions. Finally, we discuss other possible implementations and solutions with their relevant merits and pitfalls, ending with conclusions and possible future extensions.

2. BACKGROUND

There are (broadly) two flavours of surface mesh algorithms: (1) Planar Contours, that generate surface over the boundary of adjacent contour paths[3–5]; and (2) Local Cell evaluation, that can be further subdivided into: (a) Advancing Front, that finds the surface by growing a seed point on the surface, from where the other surface segments are found[6,7]; and (b) Complete Cell Evaluation, that evaluates each cell's contribution to the surface: forming a surface made from tiles.[8–10] We use and extend the latter method to four and theoretically higher dimensions. Moreover the advancing front techniques could be likewise extended to n-dimensions.

2.1. 3D Local Cell Surface Generation

The local cell tilers evaluate a single cell for its contribution to the surface. Two such methods in three dimensions are by lookup (e.g. Marching Cubes)[9] and algorithmically.[11] An estimate of the position of the surface intersection along a particular edge can be found by linear interpolation. Multiple surfaces at the same threshold can be produced by the local cell methods, but erroneous surfaces due to the locality of the surface decision (by false positives or false negatives) can be produced. Hill and Roberts[12] and Ning and Bloomethal[8] discuss some methods to disambiguate a cell and hence remove the erroneous surfaces. Degenerate triangle pieces, where the surface-simplices become infinitesimal, can also be created (as a result of the interpolation process), slowing the rendering and increasing the storage. However, decimation[13] or mesh displacement[14] techniques can be used to reduce the number of (tiny) polygons.

2.2. Surface Creation – the Use of Simplices

The local cell tilers often use a cube (rectilinear) cell representation, as in the Marching Cubes Algorithm. Tetrahedral cells have also been used,[15] the advantage being that a finer detailed surface is created and, that from local sign alternations only one surface can intersect the tetrahedron — there is no ambiguous face. However more polygons are usually generated[15] and as the tetrahedra can be divided into a local cube cell, in configurations of five or six tetrahedra, ambiguities are still present: because the isosurface is created by considering only neighbour data points. The ambiguities can be resolved using a twelve tetrahedra configuration[16] requiring an additional (tri-linear[17] or tri-cubic[18]) interpolated center point.

Simplices are also used in the representation of the surface mesh. All two and three dimensional graphic libraries support their rendering and there are algorithms that efficiently triangulate two and three dimensional areas.[19]

2.3. The Marching Cubes Surface Algorithm

A surface can intersect a cube in 256 (2^8) ways: this can be broken down into 14 cases if mirror and rotational symmetry are considered or 15 cases without the mirror operation. The 256 components can be stored in a lookup table containing appropriate surface topology segments.

The marching cubes algorithm[9] uses a binary threshold (the isosurface value) on the vertices of a cube to generate an eight bit (one for each vertex) number that is used as the key into the lookup table. The algorithm 'marches' sequentially through the data, thresholding the eight neighbouring data-samples and looking up the index to collect the surface intersections at that position. The vertices of the retrieved surface triangles are then interpolated into the position governed by the threshold value, appropriately shaded and rendered.

2.4. N-dimensional geometry

Perceiving geometry within a higher dimensional space is not intuitive. Therefore, we present some simple n-dimensional geometry facts; for more information see:.[1,20,21]

In three dimensions the rotations can be expressed as "rotations about each axis", but this does not extend to n-dimensions. There is, in fact, one rotation per pair of axes, which formulates to $N(N-1)/2$ degrees of rotation.[20] Therefore, in four dimensions there are six rotations.

There are many different projections from four dimensions to three, including: (1) Orthographic, where one coordinate can be *thrown away*; (2) Pinhole perspective, where the first $N-1$ coordinates are scaled by dividing by $(F_D - C_N)/F_L$, F_D being the Focal distance, F_L the Focal Length and C_N the nth coordinate; (3) Central, where the nth coordinate is shrunk into the $N-1$ coordinates: $(x, y, z, w) = (xF_L/(F_D - w), yF_L/(F_D - w), zF_L/(F_D - w))$; creating the popular hypercube depiction, where a cube is displayed within a cube.[20]

3. MOTIVATION AND DEFINITIONS

A lookup table, to hold a complete enumeration of the cases within three dimensions, contains 256 elements (2^n – where n is the number of vertices). Therefore, there are 65536 (2^{16}) configurations for the vertex classification on a four dimensional cube. If a Marching Cube method was applied directly to four dimensions the lookup table could become unmanageable; with an average of 20 tetrahedra (3D simplices) for each major case. Moreover techniques to subdivide the problem domain would (a) simplify the algorithm for explanation and implementation and (b) hopefully provide far more efficient storage.

Within this section we (1) present how the major-cases are generated; (2) describe a secondary partition separating the major-cases into sub-cases; and (3) describe the various transformations that are available and enumerate their respective major and sub-cases.

3.1. Major-Cases

In n-dimensions each cube has 2^n vertices each of which may be inside or outside the surface, hence the set of all possible configurations C contains 2^{2^n} elements. In Lorensen and Cline's account[9] of their local cell tiler, they identify 14 *major cases*. These correspond to sets of vertex configurations which are closed under rotation, mirror and vertex complement.

More formally, the major cases are established by partitioning C into smaller sets $C_1, C_2, \ldots, C_m$ such that: $\bigcup_{i=1}^{m} C_i = C$ i.e. the set C is covered, and $\bigcap_{i=1}^{m} C_i = \{\}$ i.e. the sets are disjoint. Most importantly, for all $e \in C_i$, C_i is a reflexive transitive closure under the one-to-many relation: $R(e) = \{(e, T_1(e)), (e, T_2(e)), \ldots, (e, T_k(e)\}$ where each T_i represents a major case invariant transformation, *e.g.* rotation, mirror or complement. Problems with surface continuity imply that complement may not be a desirable operation to include (see below).

		Dimension			
		1	2	3	4
R	Major Cases	3	6	23	496
	Sub-Cases	2	5	12	272
R,C	Major Cases	2	4	15	272
	Sub-Cases	2	3	7	99
R,M	Major Cases	2	6	22	402
	Sub-Cases	2	5	11	209
R,M,C	Major Cases	2	4	14	222
	Sub-Cases	2	3	7	74
Transformations: Rotation (R), Complement (C), Mirror (M)					

Table 1. Group sizes for the Major cases and Sub-cases

3.2. Sub-Cases

It is quite remarkable (and fortunate) that of 256 possible configurations in three dimensions, we need only consider 14 major cases (see[10] for a discussion). In three dimensions it is not too expensive to store all 256 cases. In higher dimensions, however, the number of configurations explodes. Even the number of major cases grows rapidly. It turns out that in four dimensions there are nearly as many major cases as their are configurations in three dimensions (see Table 1). In higher dimensions, the geometry associated with a major case is also more complicated. This prompts us to seek ways to reduce further the size of the tables required.

We can reduce the number of cases if we allow major cases to be constructed from a union of *sub-cases*. A sub-case represents a single fragment of boundary and is defined to be an edge-connected (both vertices have the same status with respect to the threshold) fragment of a major case.

The set of sub-cases can be computed by examining each major case and dividing it into one or more edge-connected sub-components. The sub-components can then be identified with their major case equivalents along with appropriate transformations.

In Table 1, we summarise the number of major and sub-cases in one to four dimensions. As shown in the table, the number of cases vary depending on the operators used; this is exemplified by different authors expressing the major-cases as being 14 or 15 cases, which depends on the use of the Complement operator. In Figure 1 we depict a representation of the 74 sub-cases for four dimensions.

The definition of a sub-case in this way makes some assumptions about the underlying surface, and may lead to inconsistencies in the resulting geometry (for example holes may appear). However, many other approaches lead to the same decisions being made. We return to these problems in Section 4.2.

4. IMPLEMENTATION

The n-dimensional surface is generated by connecting individual surface elements (simplices). The piecewise surface segments may be generated through lookup methods, such as the Marching cubes or algorithmically.[11] Indeed, tetrahedral, rather than cuberille cells, may be used, such as used by[22] who additionally calculate a center point in the cuberille cell.

The simplex information can be subdivided and stored in many ways. The amount and orientation of the simplices, for a given cell, represents the most significant information – being used in every method.

Further, we describe our methods to store and retrieve the simplex information:

Method 1 Dynamic Simplex Enumeration – calculate each simplex division dynamically as it is required.[11]

Method 2 Complete simplex Enumeration – pre-calculate and store each simplex division within one large table. See Figure 2.

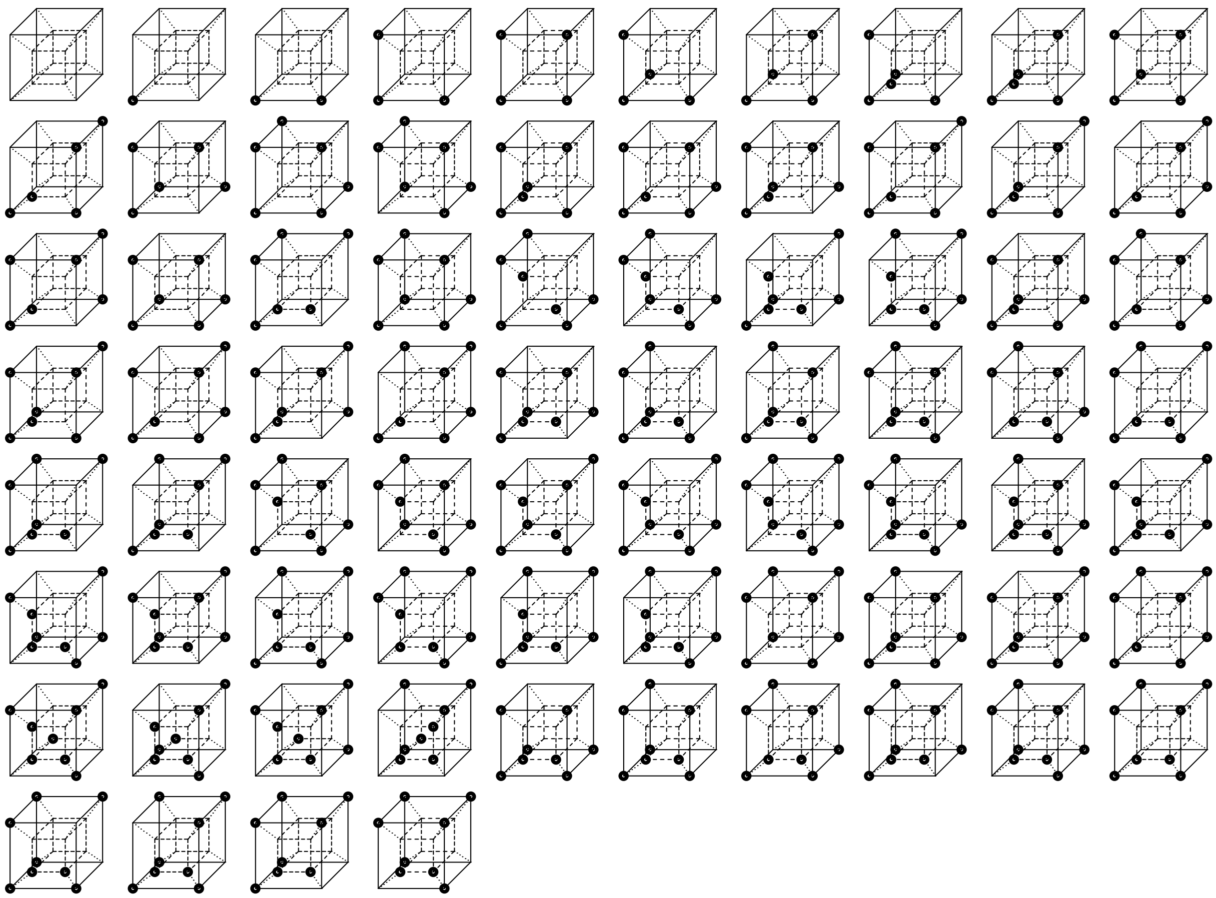

Figure 1. 74 Sub-cases for four dimensions

Method 3 Secondary Tables – pre-calculating the simplex division for just the major-cases or the sub-cases. We name these the 'Major-case secondary table' and 'Sub-case secondary table' methods, respectively.

The latter two-table lookup schemes include a primary table that holds information about either (a) the major-cases or (b) the sub-cases, with a secondary table containing the actual divisions of the major or sub-cases, respectively; as shown in Figure 2 as Method (3a) and (3b) respectively.

Within this setup, the primary table contains a list of two-tuples: an orientation with a secondary-index. The secondary-index then provides the key into a table of either major-case or sub-case simplices. The orientation represents a matrix operation, transforming the simplices (of the secondary table) into the correct orientation for the chosen surface-intersection index of the primary table.

4.1. Pre-Processing – Table Generation

We now describe how the major and sub-cases for the primary and secondary tables, respectively, are formed. The data is processed sequentially as each stage uses the results from the previous level.

The Major-cases. Two lists are created, one to hold the searched-cases and another to hold the major-cases.

The next unmarked index is taken from the search-list added to the major-case list and then exhaustively transformed into all other configurations which are marked. The process then repeats for the next major case until all the indices have been searched.

The Sub-cases. These are formed by dividing the major cases into their disjoint cases, using the edge-connected criterion (as defined in Section 3.2).

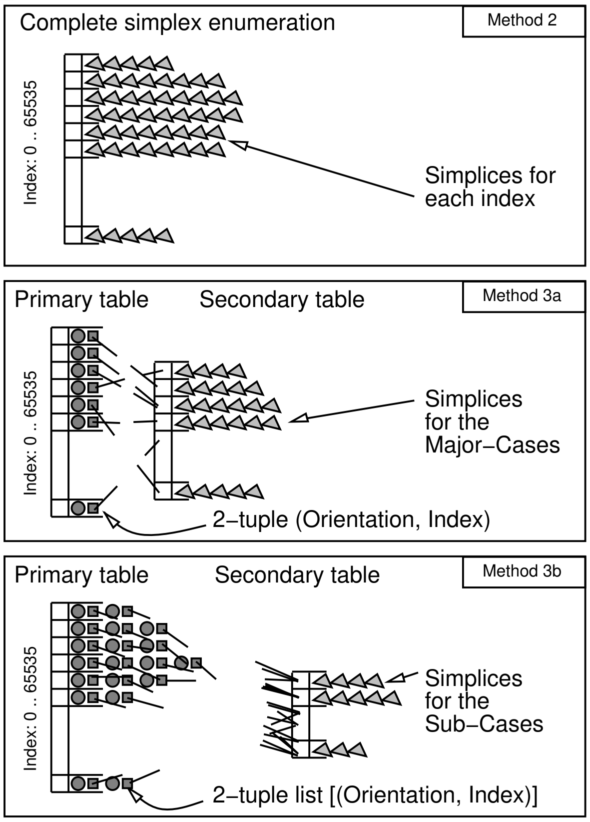

Figure 2. Table lookup schemes

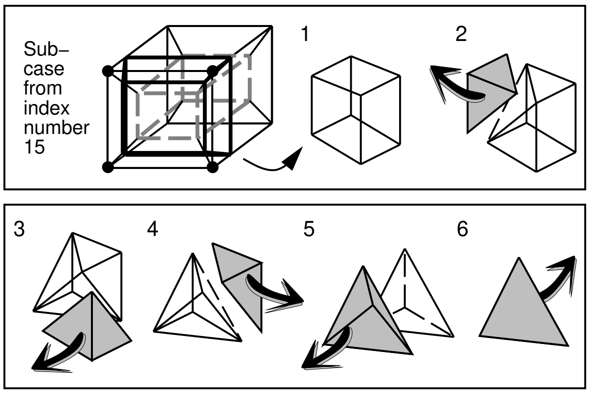

Figure 3. Splitting an object into tetrahedra

Tetrahedronizing the sub-cases. Tetrahedronization is the four-dimensional analogue of triangulation. Our aim is to split n-dimensional geometry into simplices. There is no canonical decomposition of a hyper-surface into tetrahedra, so any algorithm must make somewhat arbitrary choices.

In the context of this work, we are interested in decomposing only the sub-cases, and a simple strategy can be made to work. The method proceeds by repeatedly choosing a vertex and removing the tetrahedron associated with that vertex from the sub-case object until a single tetrahedron remains. This process is depicted in Figure 3, and is similar to the techniques used in three dimensions. Unfortunately in four dimensions, the remnant sub-object may have vertices with four, five or more incident edges making further subdivision difficult. Therefore, we use a backtracking technique to avoid this problem.

A question remains — how do we choose which vertex to remove? Several strategies might be tried: for example, choose the first entry in the vertex list or take the vertex with the least connecting edges. Experiment suggests that the latter approach is most effective in this case.

The method proceeds by:

1. The sub-case object is represented as a list of vertices, each of which is linked to a list of its neighbours. Initially, by construction, all vertices have three connections.

2. Select the next vertex (in order on the list) with the fewest incident edges, and remove it from the object thus generating a tetrahedron or two tetrahedra. It is easy to split a vertex with three edge connections into one tetrahedron or even four edge connections into two tetrahedra, but ambiguities and difficulties occur when there are five or more edge connections.

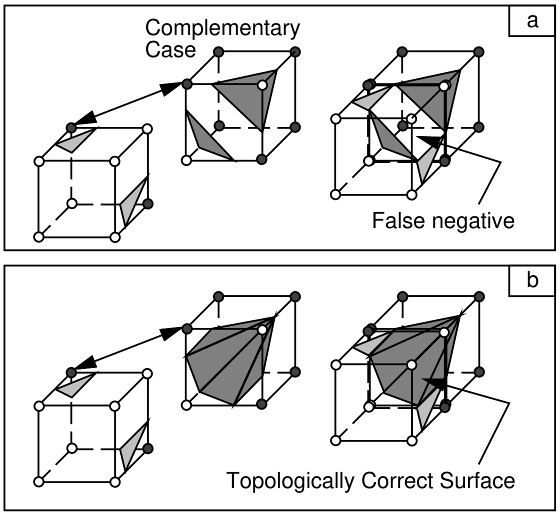

Figure 4. False negative appearing, from adjacent complementary cases

3. Update the connectivity of the remaining object. This update is achieved by connecting the vertices of the remaining object in the same configuration as the connectivity of the base of the split tetrahedron.

4. Determine if at a certain level of vertex splitting all the vertices are connected to five or more edges. If so then the previous level is reinstated and the different split attempted in 2.

In our results, using the 'fewest edges method', and only taking off the maximum of two tetrahedra at once, backtracking occurs for only three of the 74 sub-cases.

Creating the Orientation Matrix. The orientation matrix represents the transformation from the primary table (of major or sub-case indexes) into individual simplex elements and is stored as a 32 bit integer, with two bits for every position in the matrix. As all the rotations are by 90°, only values of $-1, 0, 1$ mapped onto $00, 01, 10$ respectively, are required. The orientation matrix is calculated by rotating each sub-case (from each major case) into the standard sub-case and calculating the inverse transformation matrix, of the whole operation.

4.2. Inherent Ambiguities and Possible Solutions

Ambiguities may occur in surfaces evaluated from local cell intersections. These occur in the cells when the surface intersects one face of the n-cube through each of its four edges. Therefore, as a result of local decisions spurious holes or additional surface segments can be generated. Moreover, this face occurs in two dimensions and propagates to higher dimensions, so any n-cube that has an 'ambiguous face' is potentially an ambiguous n-cube.

There are many reported disambiguation strategies for the three dimensional local cell tiling algorithms (see[12,8]) which can be divided into two groups: (1) those that provide a solution from a static analysis of the local vertices and (2) methods that require an extra sample point to generate an appropriate connected surface.

Any correct disambiguation strategy needs to be consistent, to generate a continuous connected surface. The original Marching Cubes algorithm generates an inconsistent surface when adjacent cubes of alternative (complementary) configuration are connected,[23] Figure 4a. This can be improved by individually triangulating the complementary cases,[24] Figure 4b. This configuration can be provided by an extended lookup table, with different triangle configurations for the complementary cases. In general the amount of triangle combinations required for each ambiguous face f for a given n-cube is $2^{\#f}$,[25] where $\#f$ is the number of ambiguous faces; but, in practice only a sub-set of these configurations is required,[24] being similar under rotation and generating a topologically correct surface.

Similarly, this method can be extended to n-dimensions, where the complementary cases are treated differently. Separate complementary configurations also help to maintain the vertex-order of the simplex elements: as they can be described in a clockwise order, relative to the surface-object, aiding the renderer.

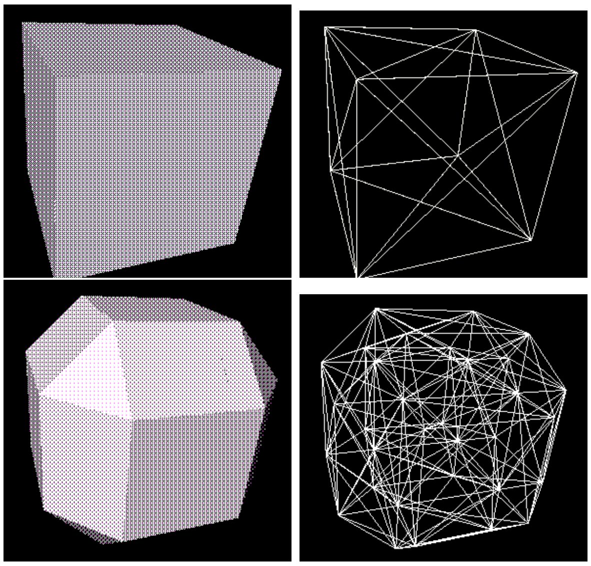

Figure 5. Examples of Application

The sub-cases are generated by separating the major cases into disjoint surface elements, the same way as in the Marching Cubes[9] and similar problems of surface continuity may result. Consequently, these sub-cases can be said, depending on the separation technique, to be ambiguous in form. Like the surface configurations in three dimensions: $2^{\#f}$ possible sub-case configurations can be formed. One simple solution is provided by using separate complementary sub-case configurations for each major case.

It can be argued that at high data-resolutions the anomalies become unobservable, although at high magnifications the anomalies could still be seen. Alternatively, a subdivision technique could be implemented: dividing the data until pixel sized cubes are formed, such as the Dividing Cubes algorithm,[10] although magnification, again, can reveal a discontinuous surface.

Other disambiguation strategies could be used and extended to higher dimensions, including using tetrahedral cells, that provide unique surface intersections (see: Section 2.2), instead of cubical cells. A dilemma occurs between the 'added advantage of the complex-disambiguation strategy' and the 'costs involved in calculating and processing the strategy'. In practice, the added computation cost is insignificant and although more simplices are generated they represent a 'small increase' on the complexity of the overall surface. Conversely, the ambiguous cases within three dimensions infrequently occur: as Neilson and Hamann[25] discovered.

5. RESULTS

Our n-dimensional surface algorithm is useful for data visualization, where the data is sampled over a rectilinear grid. Phantom data generated from analytic functions is quite easy to generate and four dimensional fractal data or the four dimensional counter-parts of the three dimensional variants can be formed – hyper-cube, cone or sphere, for example.

5.1. Examples of Application

Iris Explorer on a Silicon Graphics Indy has been used as the harness for our implementation. We have tested the algorithm on a number of sampled data sets and generated appropriate results.

Each of the methods produce the same visual results, with the same tetrahedra configurations, a simple parallel (orthographic) and perspective projection is used to generate the result. A voxel version of a four dimensional cube is displayed using the system, Figure 5. The surface intersections are generated by linear interpolation of the threshold across the edges of the cell; the upper images were generated using a low threshold, whereas the lower pair were generated using a middle threshold value.

5.2. Table Sizes

Here we describe the memory usage of each of the four methods described in section 4.

A tetrahedra contains four vertices, and each vertex can be represented by an integer label, so, each tetrahedra can be stored in 4 bytes (one byte for each vertex). Moreover, the vertices of the tetrahedra are recovered from intersections along the edges of the local cell, therefore, an alternative representation consists of a two-tuple label for each tetrahedra-vertex: relating to the edges of the hyper-cube. The former 4-byte representation will be used below.

Method 1 The dynamic method uses the least memory, but takes the longest to calculate.

Method 2 The complete simplex table consists of a 65536 array with n tetrahedrons per index. Therefore, as each tetrahedron can be stored in 4 bytes, the number of tetrahedra in 1 byte, the array pointer in 4 bytes and there are 356817 tetrahedra for the whole (222 major-case) table: the table can be stored in 1.75M bytes ($4\times356817+5\times65536$).

Method 3a The primary table for the major-cases consists of an array of 65536 (orientation, major-index) tuples: stored in (4 byte, 1 byte) portions. Therefore, the table can be stored in approximately 328K bytes (65536×5). However, many architectures may *pad* the structure to at least 6 or 8 bytes. The secondary table for the (222) major-cases consists of an array of pointers to an array of n tetrahedra, there are 2332 tetrahedrons so the table can be stored in 10.2K bytes ($4\times2332+222\times4$).

Method 3b The primary table using the sub-cases, consists of an array of 65536 pointers pointing to an array of (orientation, sub-index) tuples, stored with the size of the array – representing the number of sub-cases per major index. Each orientation and pointer can be stored in 4 byte portions. Therefore, as the whole table contains approximately 130800 sub-case indices, the table can be stored in approximately 916K bytes ($65536\times4+130800\times5$). The secondary table for the (74) sub-cases consists of 869 tetrahedra that can be stored in 3.8K bytes ($4\times869+74\times4$).

5.3. Conclusions

The complete simplex table (method 2) provides the advantage that all of the data is correctly orientated, so it can be directly applied to the data, and although the table is larger than the other methods its size is not too great (within four dimensions) to be stored on a local machine.

The primary table for the major cases is about half the size of the sub-case primary table: due mainly to the use of pointers for the 2-tuple array. The reverse is true for the size of the secondary table: where the sub-case secondary table is much smaller; we postulate that this comparison would be even more distinct in higher dimensions. However, the combined size of the primary and secondary tables falls in favour of the major-cases: due to the way the data needs to be stored.

The sub-cases, although using a fixed orientation disambiguation strategy, provide a concise representation: allowing the simplices to be calculated by a simple tetrahedronizing algorithm.

6. RELATED APPROACHES

An n-dimensional surface can be generated by many other methods. The following three parts describe: (1) other methods comparable with the hypercube approach presented herein; (2) related work; and (3) some comments on our possible future work.

6.1. Similar Techniques

A hypercube can be imagined as multiple cubes: the hypercube surface can therefore be evaluated with a series of three dimensional cubes (one for each face of the hypercube). In four dimensions a lattice of connected surfaces (rather than volumes) would be created. This lattice could then be displayed or triangulated before displaying, Figure 6B.

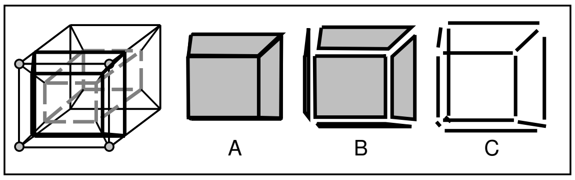

Figure 6. Cube surfaces: (A) hypervolumes, (B) hypersurfaces, (C) hyperlines

In the same manner, a two dimensional 'cube' can be evaluated through each hypercube face to create a lattice of connected hyperlines, Figure 6C. Wyvill et al[6] use the faces of a three dimensional cube (two dimensional square) to generate one dimensional lines that are joined to make surfaces in two dimensions.

An n-simplex[20,26] can be used to segment the cells unambiguously. Therefore, the n-data can be segmented directly into n-simplices. To create a uniform continuous surface the adjacent n-simplices need to be mirror images of each other, so an alternating *patchwork* of simplices are used.

An advancing front technique could be adapted into four dimensions to create a lattice of tetrahedra.[27] Hanson and Heng[28] use a similar method: they divide the whole volume into tetrahedra and then project each tetrahedron into the view volume. We use a comparable method that first selects the hypersurface part using a threshold and local cell technique, and then dissects the hypersurface into tetrahedra which are projected and rendered.

6.2. Related Work

For many years, computers have been used to generate pictures and manipulate higher dimensional objects. Noll,[29] as early as 1967, created a program to plot projected images of n-dimensional objects. He used these plots to produce a "stereo-graphic movie of the three dimensional parallel and perspective projections of four dimensional hyperobjects, rotating in four dimensional space".

Polygonising methods are also used. Bajaj[30] implicitly defines quadratic and cubic hypersurfaces in $n >= 3$ dimensional space with a constant or adaptive stepping procedure that creates a grid of points, forming polygons when connected.

Ray tracing can also be extended to higher dimensions: Ke and Panderanga[31] display projections of a four dimensional Mandelbrot set use a ray tracing technique.

Hanson and Cross[32] describe a hybrid method of ray-tracing and scan-converting to transform the four dimensional image to an equivalent three dimensional image. The problem is then reduced to a texture-mapping problem.

However, visualizations produce abstract images projected from higher dimensional data. Hanson states that "adding more visual detail may give even more clues".[33] The visual detail can be generated by: perspective projections, n-dimensional lighting, shading, object-silhouettes and colour cues within a highly manipulative environment. Therefore, greater understanding could be grasped if the data was presented by many abstract forms, within multiple directly manipulated and coupled displays, such as provided by the Waltz abstract visualization environment.[34]

Hanson and Heng[28,32] describe methods to display and shade four dimensional images using a four dimensional light model. A thickening strategy, is used, that exchanges each point on the line with a small sphere, allowing shading to be applied to the 'pseudo' line and increasing the 3D nature of the line.[28,32]

Direct manipulation techniques can also be used. Van Wijk and van Liere[35] display multidimensional scalar functions as two dimensional slices of data. The user can control any view in the matrix of windows to control the slice of each of the other views. An overall impression of the multidimensional function is obtained. Feiner and Beshers[36] have designed a model of "worlds within worlds", where three dimensional graphics are positioned within a three dimensional graph. The internal three dimensional graph changes values as it is moved inside the secondary graph by direct manipulation.

6.3. Possible Future Work

Further research should be carried out into the realism of the n-dimensional images and the understanding of four (or higher) dimensional images using visual cues gained from animation, n-dimensional shading and shadow effects.

Direct manipulation techniques could be incorporated with a dynamic version of the algorithm, to achieve rotation through the fourth and higher dimensions. Keeping a cache of the last n-dimensional indices, tetrahedronized sub-cases and major case elements might be appropriate.

7. DISCUSSION

We have explorered the possibility of extending an isosurface technique to n-dimensions and implemented some algorithms for four dimensions. The research has produced: (1) methods to enumerate major and sub-cases; (2) a simple and effective tetrahedronizing algorithm for the sub-cases; and (3) a method for major cases using sub-cases and a compact orientation method.

Local cell evaluation is ambiguous in n-dimensions: generating incorrectly placed surface simplices. The ambiguities arise in two dimensions and propagate to higher dimensions; they can be disambiguated using algorithms equivalent to the three dimensional strategies.

Other n-dimensional surface methods were proposed, including, generating hypersurfaces or hyperlines from the higher dimensional data and marching hypertetrahedra directly through the higher dimensional space. A connected surface is generated when adjacent simplices are placed in an alternating configuration.

The table lookup is extensible to n-dimensions. However, as the dimensions increase, so the size of the table increases; even with the sub-cases the size of the table at (say) five dimensions (2^{32}) would become impracticable and the time to calculate the major and sub-cases would be lengthy. Within four dimensions the advantage in using the lookup table over and algorithmic method is slight: our experiments suggest that there is no significant speed difference between each method. Algorithmic surface evaluation methods are probably most appropriate for higher dimensions.

The sub-cases, albeit using a fixed disambiguation strategy, allow the hyper-surface segments to be tetrahedronized by a simple algorithm.

Finally, the sub-cases provide an elegant method to split up the major cases, to easily describe the individual cases and to reduce the number of cases in the higher dimensions.

REFERENCES

1. T. F. Banchoff, *Beyond the third Dimension: Geometry, computer graphics and higher dimensions*, Scientific American Library, 1990.
2. A. Inselberg and B. Dimsdale, "Parallel coordinates: a tool for visualizing multi-dimensional geometry," in *Proceedings Visualization '90 – sponsored by the IEEE Computer Society*, pp. 361–378, 1990.
3. H. Fuchs, Z. M. Kedmen, and S. P. Uselton, "Optimal surface reconstruction for planar contours," *Communications of the ACM* **20**(10), pp. 693–702, 1977.
4. S. Ganapathy and T. G. Dennehy, "A new general triangulation method for planar contours," *Computer Graphics SIGGRAPH Proceedings* **16**, pp. 69–75, 1982.
5. D. Meyers, S. Skinner, and K. Sloan, "Surfaces from contours," *ACM transactions on Graphics* **11**(3), pp. 228–258, 1992.
6. G. Wyvill, C. McPheeters, and B. Wyvill, "Data structure for soft objects," *The Visual Computer* **2**(4), pp. 227–234, 1986.
7. A. Norton, "Generation and display of geometric fractals in 3-D," *ACM Computer Graphics* **16**, pp. 163–169, July 1982.
8. P. Ning and J. Bloomenthal, "An evaluation of implicit surface tilers," *IEEE Computer Graphics and Applications* **13**, pp. 33–41, November 1993.
9. W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM Computer Graphics* **21**, pp. 163–169, July 1987.
10. 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**, pp. 320–327, May/June 1988.

11. J. Bloomenthal, "An implicit surface polygonizer," in *Graphics Gems IV*, P. S. Heckbert, ed., pp. 324–349, Academic Press, 1994.
12. S. Hill and J. C. Roberts, "Surface models and the resolution of N-Dimensional cell ambiguity," in *Graphics Gems V*, A. W. Paeth, ed., pp. 98–106, Academic Press, 1995.
13. W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," *Computer Graphics SIGGRAPH Proceedings* **26**, pp. 65–70, July 1992.
14. D. Moore and J. Warren, "Mesh displacement: An improved contouring method for trivariate data," Tech. Rep. 91-166, Rice University, Department of Computer Science, December 1991. (dougm@kuhn.caam.rice.edu, jwarren@cs.rice.edu).
15. C. Zahlten, "Piecewise linear approximation of isovalued surfaces," in *Advances in Scientific Visualization*, F. H. Post and A. J. S. Hin, eds., pp. 105–118, Springer-Verlag, 1992.
16. J. Bloomenthal, "Polygonization of implicit surfaces," *Computer Aided Geometric Design* **5**, pp. 341–355, 1988.
17. S. Hill, "Tri-linear interpolation," in *Graphics Gems IV*, P. Heckbert, ed., pp. 521–525, Academic Press, 1994.
18. L. Arata, "Tri-cubic interpolation," in *Graphics Gems V*, A. W. Paeth, ed., pp. 107–110, Academic Press, 1995.
19. M. Bern and D. Eppstein, "Mesh generation and optimal triangulation," in *Computing In Euclidean Geometry*, D.-Z. Du and F. Hwang, eds., pp. 23–90, World Scientific, 1992.
20. A. J. Hanson, "Geometry of n-dimensional graphics," in *Graphics Gems IV*, P. S. Heckbert, ed., pp. 149–170, Academic Press, 1994.
21. D. M. Y. Sommerville, *An introduction to the geometry of n dimensions*, New York, Dover Publications, 1879. ref'd in ND geometry paper.
22. C. Weigle and D. C. Banks, "Complex-valued contour meshing," in *IEEE Visualization '96*, pp. 173–180, IEEE, Oct. 1996. ISBN 0-89791-864-9.
23. M. J. Duurst, "Additional reference to marching cubes," *Computer Graphics* **22**, pp. 72–73, April 1988.
24. W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics*, Prentice-Hall, 1996.
25. G. M. Neilson and B. Hamann, "The asymptotic decider: Resolving the ambiguity in the marching cubes," in *Proceedings Visualization '91 – sponsored by the IEEE Computer Society*, pp. 83–91, 1991.
26. D. Moore, "Understanding simploids," in *Graphics Gems III*, D. Kirk, ed., pp. 250–255, Academic Press, 1992.
27. H. Jin and R. I. Tanner, "Generation of unstructured tetrahedral meshes by advancing front technique," *International Journal for Numerical Methods in Engineering* **36**, pp. 1805–1823, 1993.
28. A. J. Hanson and P. A. Heng, "Illuminating the fourth dimension," *IEEE Computer Graphics and Applications* **12**, pp. 54–62, July 1992.
29. M. A. Noll, "A computer technique for displaying n-Dimensional hyperobjects," *Communications of the ACM* **10**, pp. 469–473, August 1967.
30. C. L. Bajaj, "Rational hypersurface display," *Computer Graphics, ACM SIGGRAPH* **24**, pp. 117–127, March 1990.
31. Y. Ke and E. S. Panduranga, "A journey into the fourth dimension," in *Proceedings Visualization '90 – sponsored by the IEEE Computer Society*, pp. 219–229, 1990.
32. A. J. Hanson and R. A. Cross, "Interactive visualization methods for four dimensions," in *Proceedings Visualization '93 – sponsored by the IEEE Computer Society*, pp. 196–203, 1993.
33. G. Zorpette, "Graphics — a romance with many dimensions," *IEEE Spectrum* **30**, p. 18 and 86, January 1993.
34. J. C. Roberts, *Aspects of Abstraction in Scientific Visualization.* Ph.D thesis, University of Kent at Canterbury, Computing Laboratory, Canterbury, Kent, England, UK, CT2 7NF, October 1995.
35. J. J. van Wijk and R. van Liere, "HyperSlice – visualization of scalar functions of many variables," in *Proceedings Visualization '93 – sponsored by the IEEE Computer Society*, pp. 119–125, 1993.
36. S. Feiner and C. Beshers, "Visualizing n-Dimensional virtual worlds with n-vision," *Computer Graphics, ACM SIGGRAPH* **24**, pp. 37–38, March 1990.