www.cadanda.com
Computer-Aided Design AND APPLICATIONS™

Weaving a Four-dimensional Mesh Model from a Series of Three-dimensional Voxel Models

Masahiko Onosato[1], Yosuke Saito[2], Fumiki Tanaka[3] and Ryoji Kawagishi[4]

[1]Hokkaido University, onosato@ssi.ist.hokudai.ac.jp
[2]Hokkaido University, saito@dse.ssi.ist.hokudai.ac.jp
[3]Hokkaido University, ftanaka@ssi.ist.hokudai.ac.jp
[4]FOR-A Co. Ltd.

ABSTRACT

This paper presents a method of four-dimensional (4-D) mesh modeling that generates a 4-D mesh model from a series of three-dimensional (3-D) voxel models. To reduce the size of model data and the computing time for mesh generation, the authors have been developing 4-D Mesh Weaver which generates a 4-D mesh model in a compact data model by incrementally loading 3-D voxel files. For efficient processing in 4-D Mesh Weaver, a 4-D triangulation table was preprocessed for fast processing, and an algorithm of 4-D mesh generation was improved so as to avoid the redundant vertices of mesh tetrahedrons.

Keywords: four-dimensional model, tetrahedron mesh, voxel, marching hyper-cube.

1. INTRODUCTION

Four-dimensional (4-D) modeling for representing and analyzing a dynamic object which continuously changes its shape and position over time is now required in various fields such as robotics [5], manufacturing [8], building construction [8], medical science [4] and so on. The much-used way for 4-D representations is to make a series of three-dimensional (3-D) frames in which objects at specific time points are represented as 3-D geometric models. This approach to 4-D modeling based on conventional 3-D modeling divides a continuous spatio-temporal space into cross-sectional 3-D subspaces, each of which represents an object's scene with no time variations. The behavior of a dynamic object is discretely recorded as a series of static 3-D frames as well as movie films and animation videos. To evaluate time-dependent properties of the object like motion and deformation, we should retrieve the necessary scenes at particular time points and then reproduce the changes of the object from its multiple 3-D models. Thus, the time axis in a spatio-temporal space is treated differently than other three axes of a 3-D subspace and the way of evaluating time-dependent properties is deeply dependent on the programming of model data handling.

Four-dimensional geometric modeling is introduced to represent a dynamic object in a 3-D space as a static object in a 4-D space. The time axis is continuously dealt with in the same way as the other three axes. The authors' research group in Hokkaido University has been studying 4-D geometric modeling for representing a dynamic object in mechanical design and manufacturing from 2007. Our first step of 4-D geometric modeling is to develop a 4-D mesh modeling system with which the results of 3-D motion and deformation simulation are continuously recorded as a 4-D geometric model. Fig. 1 shows the configuration of the 4-D mesh modeling system developed by the authors' group and named 4-D Mesh Modeler. This paper mainly focuses on the process of 4-D mesh weaving in 4-D Mesh Modeler shown in Fig. 1. This process has been implemented as 4-D Mesh Weaver, one of the subsystems of 4-D Mesh Modeler. With 4-D Mesh Weaver, 3-D voxel models captured from a 3-D simulator are incrementally translated into a continuous 4-D mesh model.

This paper firstly provides a brief introduction of 4-D geometry and tetrahedron mesh representation in the next section, then makes an explanation of Bhaniramka's isosurfacing method [2] using 4-D voxels in Section 3. In Section 4, the main part of this

Computer-Aided Design & Applications, 11(6), 2014, 649–658, http://dx.doi.org/10.1080/16864360.2014.914383

Taylor & Francis

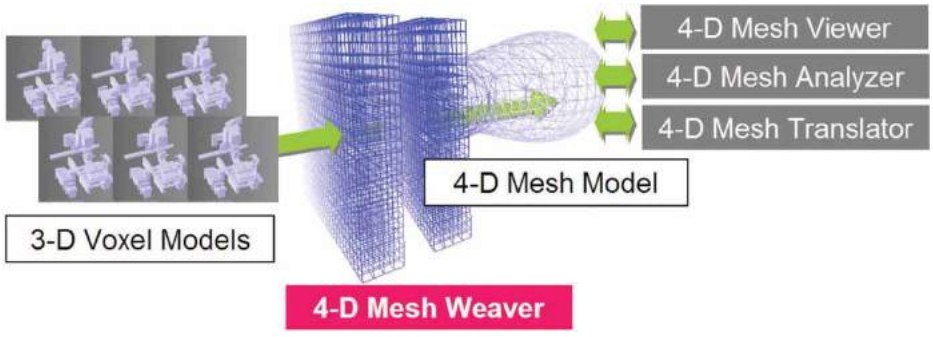

Fig. 1: Concept of 4-D Mesh Weaver in 4-D geometric modeling.

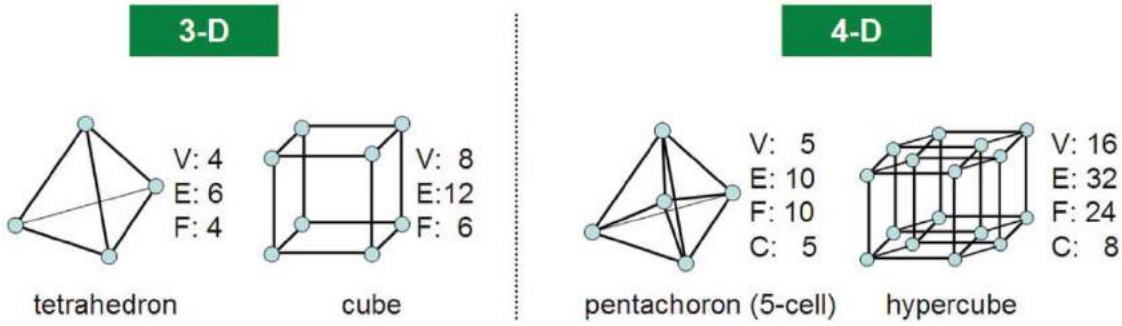

Fig. 2: Simplex and cube in 3-D and 4-D space.

paper, an efficient method in 4-D mesh weaving and its implementation of 4-D Mesh Weaver are explained with 4-D mesh model examples. Section 5 provides some issues related to 4-D mesh modeling, followed by conclusions in Section 6.

2. FOUR-DIMENSIONAL MESH MODEL

2.1. Geometrical Elements in Four-dimensional Space

As an introduction of 4-D geometric modeling, let us start with explanation of geometrical elements in 4-D space. The simplest shape in 3-D space (3-simplex) is a tetrahedron, which has 4 vertices (V), 6 edges (E), and 4 faces (F). A cube is the most popular polyhedron in 3-D space. Every polyhedron in 3-D space satisfies the equation $V - E + F = 2$. (See Fig. 2 (a).) In 4-D space, 4-simplex is a pentachoron, which has 5 vertices, 10 edges, 10 faces, and 5 cells (C) as shown in Fig. 2 (b). The four-dimensional analogue of a cube is a hypercube, which has 16 vertices, 32 edges, 24 square faces, and 8 cells. As well as 3-D space, every polychoron in 4-D space satisfies the equation $V - E + F - C = 0$.

Each polychoron in 4-D space is bounded by cells, as well as each polyhedron in 3-D space is bounded by faces. Cells in 4-D space are polyhedrons on 3-D subspaces (hyperplanes), and they can be decomposed into tetrahedrons. Thus, every polychoron in 4-D space is represented by a set of tetrahedrons on its boundary, which is called 4-D mesh model in this study.

Four points placed in 4-D space $p_i = (x_i, y_i, z_i, t_i) \in \mathbf{R}^4, (i = 0, \ldots, 3)$ are formed a tetrahedron on a unique hyperplane which normal vector $\boldsymbol{n}$ is calculated by normalizing following 4-D vector $\boldsymbol{m}$ as $\boldsymbol{n} = \boldsymbol{m}/|\boldsymbol{m}|$.

$$\boldsymbol{m} = \begin{vmatrix} \mathbf{e}_x & \mathbf{e}_y & \mathbf{e}_z & \mathbf{e}_t \\ x_1 - x_0 & y_1 - y_0 & z_1 - z_0 & t_1 - t_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 & t_2 - t_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 & t_3 - t_0 \end{vmatrix} \tag{2.1}$$

where $\boldsymbol{e}_x$, $\boldsymbol{e}_y$, $\boldsymbol{e}_z$ and $\boldsymbol{e}_t$ are unit vectors along x, y, z and t axes respectively.

To determine which half space is inside or outside of each boundary tetrahedron consistently, the serialization of tetrahedron's four vertices is very important. For example, index order 0-1-3-2 reverses a normal vector of index order 0-1-2-3. Such an important note on the normal vector direction of tetrahedron elements in 4-D space is just similar to the distinction of CW/CCW loops of triangles in 3-D mesh modeling.

2.2. Data Structure of Four-dimensional Mesh Models

Four-dimensional mesh modeling needs more geometric elements than ones in 3-D mesh modeling. To understand higher dimensions increase the number of geometric elements for model representation, a simple example of a 4-D model is explained here.

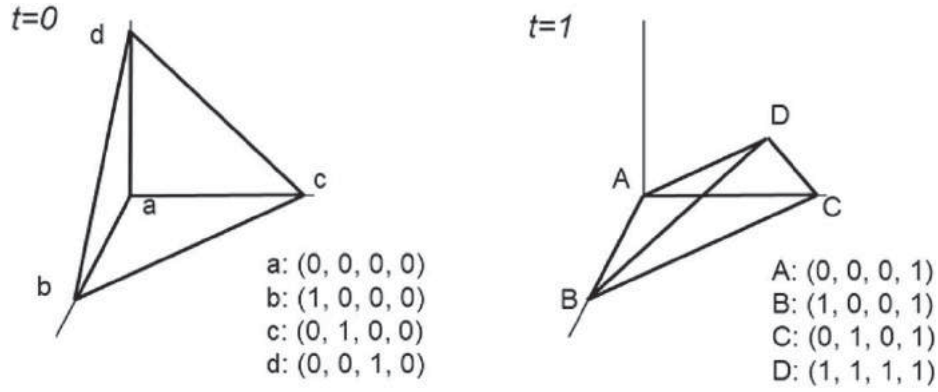

Fig. 3: A simple example of a 4-D modeling: tetrahedron's deformation.

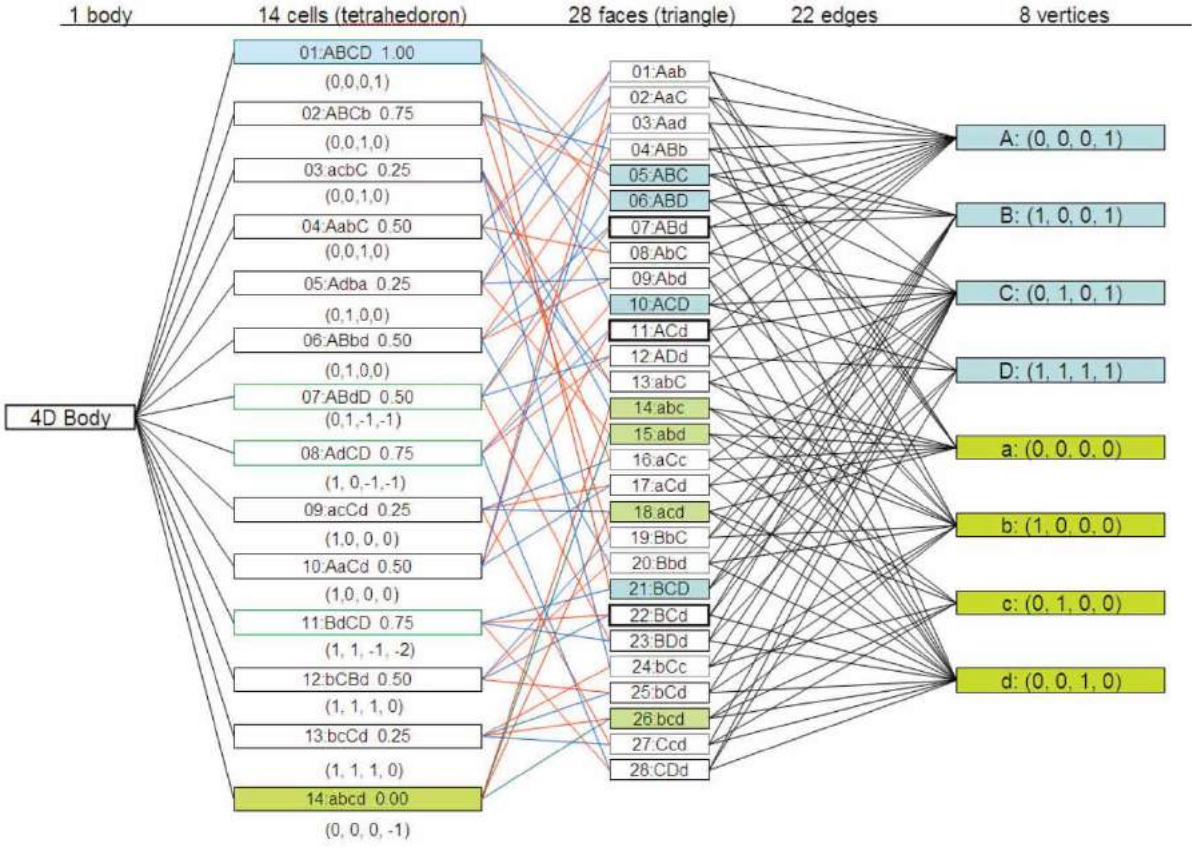

Fig. 4: Geometric data structure of a 4-D mesh model for tetrahedron's deformation.

An example shown in Fig. 3 is a deforming tetrahedron. At time 0, four vertices the tetrahedron are at 4-D coordinate a, b, c and d respectively. At time 1, three points a, b and c stay at the same x, y and z coordinates as time 0, but d (0, 0, 1, 0) is continuously moving to point D (1, 1, 1, 1) shown in Fig. 3. Consequently the tetrahedron is deformed during time 0 and 1. Then, let us see how the deformation process is represented in 4-D mesh modeling.

Fig. 4 shows geometric elements and their relations defined in the 4-D mesh model of a deforming tetrahedron. All vertices in the 4-D model come from original tetrahedron's vertices; a, b, c, d, A, B, C, and D. There are 14 cells that determine the boundary of the 4-D body. Two of them are original tetrahedrons abcd and ABCD and the others are newly generated during time 0 and 1. Each tetrahedron has four adjacent tetrahedrons and shares its triangle faces with them. The numbers of triangle faces and edges are 28 and 22 respectively.

The complicated data structure of a 4-D mesh model shown in Fig. 4 is prepared to demonstrate

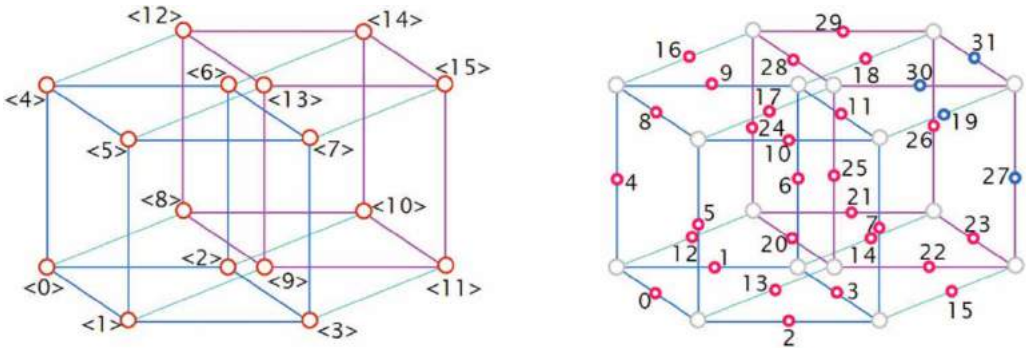

Fig. 5: Hypercube's vertex index numbers (left) and edge index numbers (right).

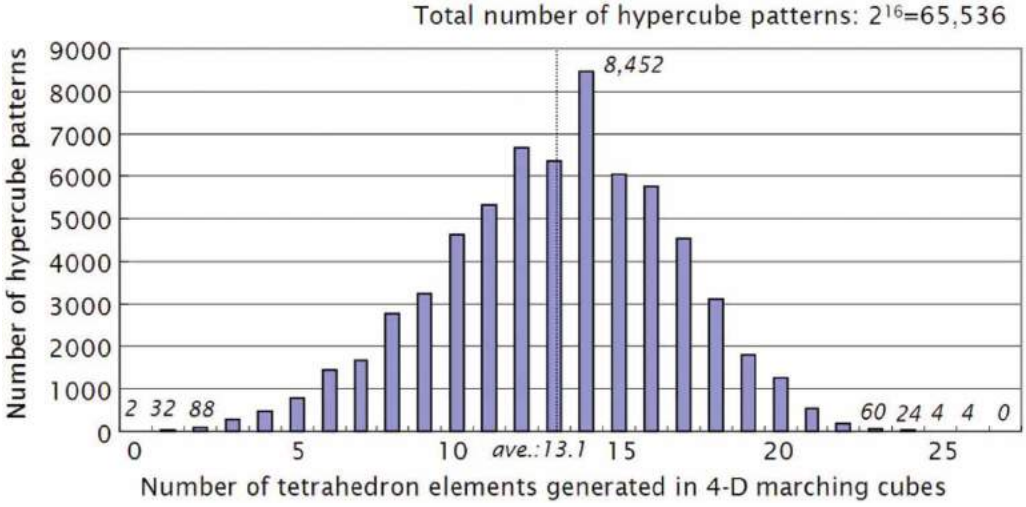

Fig. 6: Distribution of tetrahedron elements defined in 4-D triangulation table.

how many geometric entities and relations are necessary for representing a simple 4-D body. To reduce the data size of 4-D mesh models, we can omit faces and edges from 4-D model structure since these entities and relations are uniquely derived from data of cells. Information about faces and edges are efficiently retrieved by introducing neighborhood links that connect adjacent tetrahedrons.

3. ISOSURFACING A FOUR-DIMENSIONAL VOXEL MODEL

One of the most feasible ways to construct 4-D mesh models is to do isosurfacing 4-D voxel data proposed by Bhaniramka, et al. [2] in 2000. Bhaniramka's method extends the marching cubes for 3-D space by Lorensen et al. [9] to 4-D space. The 4-D marching cubes method translates 4-D voxel data to a set of tetrahedrons by looking up the 4-D triangulation table for 2^{16} patterns of hypercubes. Each pattern is identified by 16 bits of inside/outside flags and it means one of numbers from 0 (all voxels are outside) to 65,535 (all voxels are inside). All vertices of tetrahedrons generated in the 4-D marching cubes method are located on edges of a hypercube and identified by using edge index number as shown in Fig. 5.

The 4-D triangulation table provides a mapping between hypercube pattern codes and a set of tetrahedrons to be generated. For example, a hypercube pattern code 10 ($2^3 + 2^1$), which means vertex No. 1 and 3 have an inside flag and other vertices are all outside, has three tetrahedron patterns which edge numbers are 15-3-7-13, 7-3-5-13, and 3-0-5-13. This mapping table contains 65,536 hypercube patterns and 856,960 tetrahedrons that are classified into 8,036 vertex-on-edge patterns. As shown in Fig. 6, hypercube patterns in 4-D triangulation table are translated to an average of 13.1 tetrahedron elements, and at most, 26 tetrahedrons. Thus, the 4-D marching cubes method usually outputs a large number of tetrahedrons from 4-D voxel data and this makes it difficult to apply this approach to a large-scale time-varying data [10].

4. FOUR-DIMENSIONAL MESH WEAVER

4.1. Problems of 4-D Mesh Generation from 4-D Voxel Data

Four-dimensional mesh generation in 4-D space by marching cubes is almost the same as one in 3-D space except for its large data size. To make a 4-D mesh modeling system executable on a desktop PC with good interactivity, it is necessary to reduce both memory size and computing time used for 4-D mesh generation. Main problems to be solved are as follows;

- **Big data size of 4-D voxel:** The size of 4-D voxel data easily exceeds the capacity of a computer when you want to make a long-term model or precise time-step model. While the data size of 1024^3 voxels and 1B/voxel in 3-D space is about 1GB, a 4-D voxel model extended for 1024 time-steps requires 1TB of memory.
- **Large 4-D triangulation table:** As described in Section 3, the 4-D triangulation table contains 856,960 tetrahedrons for 65,536 hypercube patterns. The size of its ASCII file is approximately 9.4MB and it requires almost the same size of memory when the file is loaded. Although the size is not so large compared with 4-D voxel and 4-D mesh model, a smaller sized table is preferable for parallel processing by GPGPU.
- **Time-consuming process of merging redundant vertices:** When tetrahedron generation in 4-D marching cubes is executed on each hypercube independently, vertices of generated tetrahedrons are newly created on edges of each hypercube. Since every edge of a hypercube is shared with other seven neighbor hypercubes, eight vertices may be created on the same edge in multiple. Merging redundant vertices is a necessary process to evaluate the adjacency of tetrahedrons and to reduce the memory space for a 4-D mesh model, but it is the mostly time-consuming process in 4-D mesh generation.
- **Enlarged topological structure:** To add a dimension to 3-D modeling makes geometric model structure of boundary representations more complicated due to the introduction of a new topological element, cell, and increase data reference links between topological elements as shown in Fig. 4.

To resolve or avoid these problems, the authors' group has been implementing a lean mesh translator named 4-D Mesh Weaver.

4.2. Incremental Processing of a Series of 3-D Voxel Models

As described in Subsection 4.1, 4-D voxel data are too large to be allocated on the main memory in whole. Most applications of 4-D mesh modeling are related to time-variation processes and produce a time series of 3-D voxel data incrementally. To prepare hypercubes of inside/outside patterns for tetrahedron mesh generation, at least two successive 3-D voxel models should be loaded on the main memory. The 4-D mesh data generated from hypercubes can be incrementally added to model data file, and then cleared from the memory. This step-by-step mesh generation of 4-D Mesh Weaver, as well as incremental generation in 3-D marching cube[9] ,enables to reduce the size of main memory and build a 4-D mesh model of many time steps.

As a part of the 4-D mesh modeling system, we have been developing a capturing tool for a series of 3-D voxel models. This tool uses OpenGL's API hook and captures OpenGL commands of scene definition from a 3-D application such as simulator. Captured commands of OpenGL are executed on a graphic board and 3-D voxel data are prepared on the graphics memory of the board by using Eisemann's voxelization method [6]. This approach eliminates the need of preparing large 3-D voxel data files and has a high compatibility with parallel processing on GPGPU. In this paper, we won't go into a detail of this 3-D voxel data capturing tool using API hook and a graphic board.

4.3. Preprocessing of a 4-D Triangulation Table

The 4-D triangulation table is the most important part in 4-D mesh generation, and it is referred by each hypercube to decide tetrahedron patterns to be generated. We analyzed the content of the original 4-D triangulation table that consists of tuples of a hypercube pattern code, the number of tetrahedrons, and vertex index sequences, and then designed a data structure of the 4-D triangulation table so as to reduce memory space and operation steps.

The data structure of the 4-D triangulation table is illustrated in Fig. 7. The original table is converted to two tables and a list: the hypercube pattern table, the tetrahedron list, and the edge list table. The hypercube pattern table is an entry table that gives the number of tetrahedrons, new vertex flag code, and start index of the tetrahedron list. Each vertex flag code has information of vertices on a hypercube to be created. As explained later, possible new vertices in a hypercube are No. 19, 27, 30, and 31 in Fig. 5 (right) and 4 bits in 1 byte are used as flags of "created" or "not created". The tetrahedron list just stores tetrahedron indices referred from the hypercube pattern table. This list needs large memory for more than 8.5×10^5 tetrahedrons. To make this list compact, we introduce tetrahedron pattern indices. We have analyzed tetrahedron patterns of a hypercube in 4-D marching cubes and assigned an index to each pattern. By using tetrahedron pattern indices instead of lists of tetrahedron vertices code, the size of required

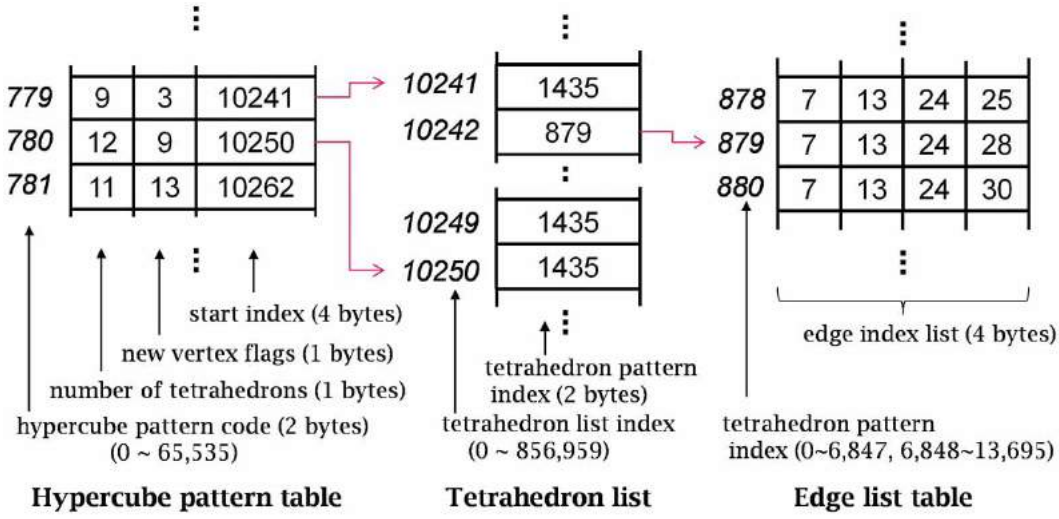

Fig. 7: Data structure of 4-D triangulation table preprocessed for efficiency.

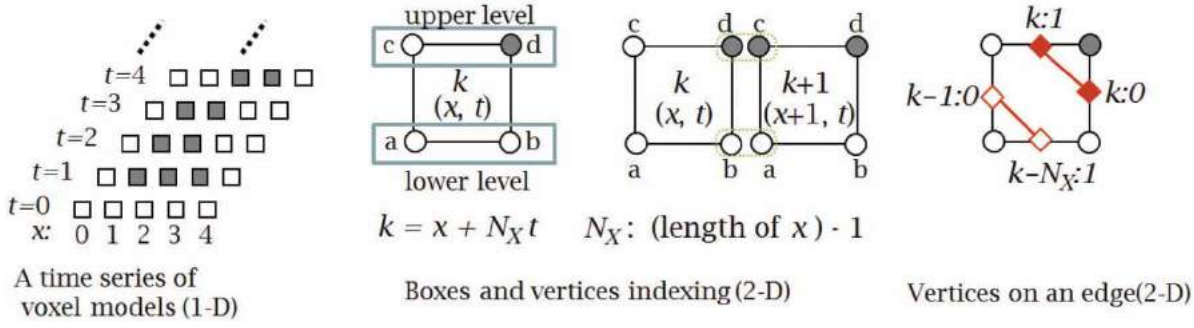

Fig. 8: Indexing concept of boxes and vertices in a simplified 2-D example.

memory is reduced to almost one-third. The edge list table is used to get a list of tetrahedron's vertices generated on hypercube edges from tetrahedron pattern indices.

4.4. Advancing Frontier Edge Algorithm of 4-D Marching Cubes

As described in Subsection 4.1, multiply created vertices on hypercubes require larger memories for temporal working space and long computation time for merging vertices. Based on 3-D approach[11], the authors have developed an advancing frontier edge algorithm of 4-D marching cubes by which the vertex merge process is not necessary since each vertex is created just one-time. To explain this algorithm, a simplified example of mesh generation from a series of one-dimensional voxel models is introduced and illustrated in Fig. 8 and Fig. 9.

A time series of 1-D voxel (vector) models that stores 0/1 values is loaded to a main memory one-by-one. The first voxel model, $t = 0$, is prepared for model-end termination and its values are set to all zero. When the length of voxel model is $N_x + 1$, N_x boxes are prepared and set in line. Each box has its coordinate values (x, t) and index value calculated as $k = x + N_x t$. Each box has four vertices a, b, c and d as shown in Fig. 9. The edges c-d and b-d on the box are named advancing frontier edges since these edges have no adjacent boxes as processed. As shown in Fig. 8, each box has two levels; the lower level with vertices a, b and the upper level with vertices c, d. The lower level vertices are holding values of the previous voxel model, while the upper level is holding the current model. A voxel value of the current model is set to each vertex of two adjacent boxes. In Fig. 8, vertex d on the box indexed k has the same value with vertex c on the box indexed $k + 1$. To distinguish the vertices created on edges, a notation of (box index): (edge index) is introduced. A vertex on the right edge has the edge index of 0, and one on the top edge has 1. The left edge index is also 0 as well as the right edge, but we use the left box index. A vertex on the bottom edge is referred by the box index of the previous layer. This indexing method enables duplication-free creation of vertices. Fig. 9 shows the process of line generation based on the advancing edge frontier algorithm. When the

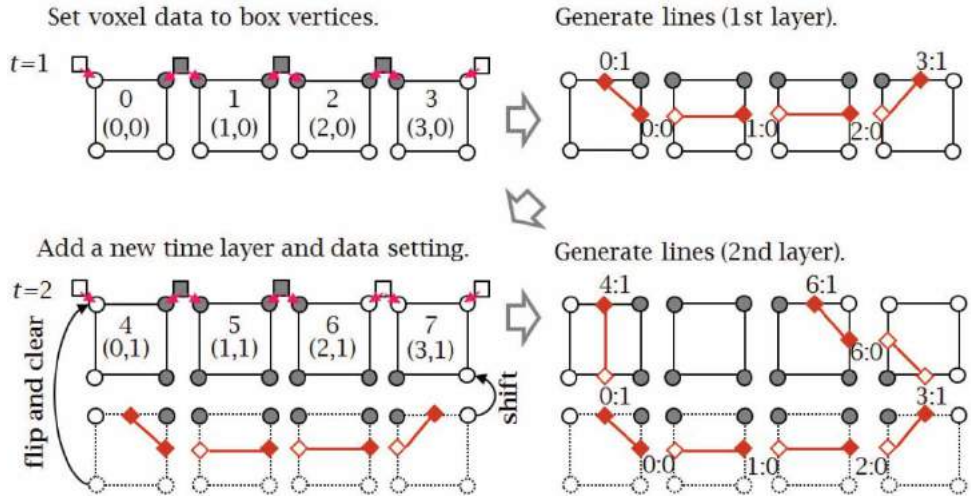

Fig. 9: Line generation process in a simplified 2-D example.

Edge	*dx*	*dy*	*dz*	*dt*	*Pn*	Edge	*dx*	*dy*	*dz*	*dt*	*Pn*	Edge	*dx*	*dy*	*dz*	*dt*	*Pn*
0	0	−1	−1	−1	3	12	−1	−1	−1	0	0	20	0	−1	−1	0	3
1	−1	0	−1	−1	2	13	0	−1	−1	0	0	21	−1	0	−1	0	2
2	0	0	−1	−1	2	14	−1	0	−1	0	0	22	0	0	−1	0	2
3	0	0	−1	−1	3	15	0	0	−1	0	0	23	0	0	−1	0	3
4	−1	−1	0	−1	1	16	−1	−1	0	0	0	24	−1	−1	0	0	1
5	0	−1	0	−1	1	17	0	−1	0	0	0	25	0	−1	0	0	1
6	−1	0	0	−1	1	18	−1	0	0	0	0	26	−1	0	0	0	1
7	0	0	0	−1	1	19	0	0	0	0	**0**	27	0	0	0	0	**1**
8	0	−1	0	−1	3							28	0	−1	0	0	3
9	−1	0	0	−1	2							29	−1	0	0	0	2
10	0	0	0	−1	2							30	0	0	0	0	**2**
11	0	0	0	−1	3							31	0	0	0	0	**3**

Tab. 1: Hypercube edge numbers and offset parameters for hypercube index coding.

line generation process for one layer is completed, the vertices on the upper level and the lower level are swapped, and the upper level is cleared for new layer setting. By swapping the two levels and reusing them, the space of main memory is necessary only for one layer.

This algorithm explained in 2-D space can be naturally extended to 4-D space as the advancing frontier edge algorithm for 4-D marching hypercubes. In 4-D space, there are four advancing frontier edges in a hypercube as shown in Fig. 5 (right). They are No. 19, 27, 30 and 31, and referred as 0, 1, 2 and 3 respectively. Each hypercube has a coordinate (x, y, z, t) in 4-D space and its hypercube index value is defined as $k = x + N_x(y + N_y(z + N_z t))$, where N_x, N_y, N_z are the numbers of hypercubes along x, y, and z respectively. A vertex created on an edge of a hypercube is noted as the combination of a hypercube index code k and an edge identifier *Pn* as *k*: *Pn* as defined in following formula (4.1) and Tab.1.

$$k = (x + dx + N_x(y + dy + N_y(z + dz + N_z(t + dt)))) \tag{4.1}$$

Created vertices notated in k : *Pn* form are sorted in a vertex list and each of them is assigned an index number of the vertex list.

4.5. Compact Data Structure for 4-D Mesh Models

Four-dimensional geometric modeling easily increases the amount of models data, and much care of model data reduction is needed for software implementation. Because the 4-D mesh modeling deals with only tetrahedrons as boundary elements in 4-D space, the authors have designed a compact data structure for 4-D mesh models. This data structure represents essential elements and relations for 4-D mesh models and additional attributes can be added to the structure for the use of applications or for faster calculation. The data structure for 4-D mesh model consists of three object classes: 4-D body, Tetrahedron, and Vertex as shown in Fig. 10.

The class Tetrahedron is the key object in the 4-D mesh modeling. It holds a sequence of four vertices and a sequence of four neighbor tetrahedrons. The

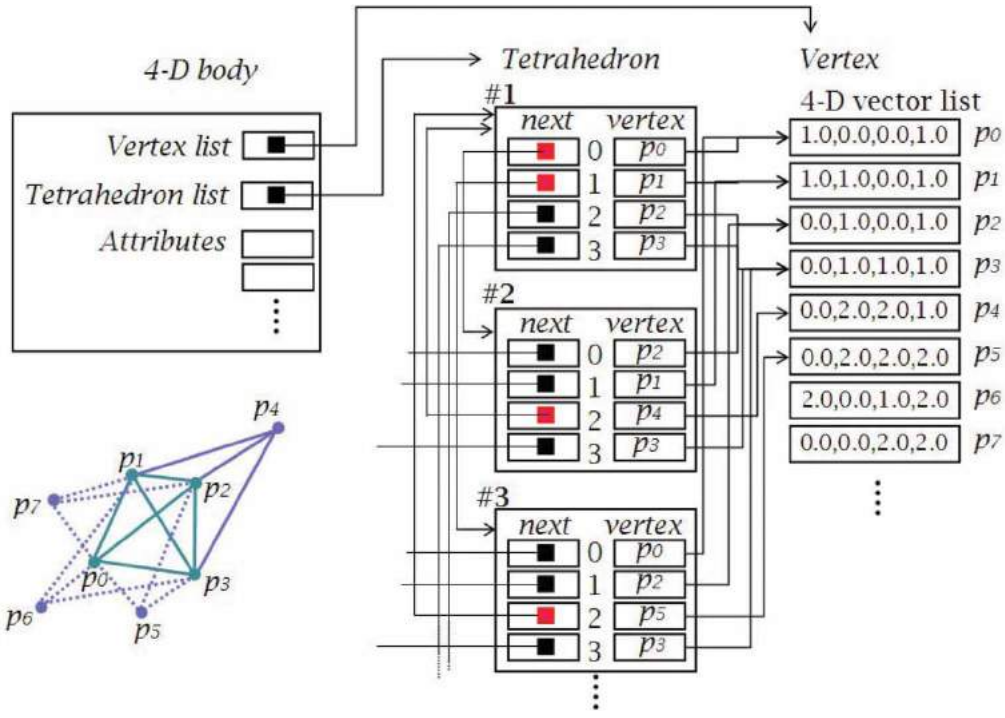

Fig. 10: Compact data structure of 4D tetrahedron for representing connection network.

direction of the normal vector in a tetrahedron is determined by the order of vertices as described in Subsection 2.1. Every tetrahedron shares its triangle faces with its neighbor tetrahedrons. In the case of Fig. 10, tetrahedron #1 is connected by next link 0 to tetrahedron #2 via mating triangle face $p1 - p2 - p3$. Mating triangle faces are identified by a vertex not included in the triangle face. Therefore, triangle face $p1 - p2 - p3$ is uniquely identified by the excluded vertex $p0$ and the index of its next link is used as the same index number as the excluded vertex. On the other hand, tetrahedron #2 is connected to tetrahedron #1 with next link 2 by mating triangle face $p3 - p2 - p1$.

An object of the 4-D body class keeps a list of vertices, a list of tetrahedrons, and other attribute items. In addition to the compact data structure based on tetrahedron objects, a tree structure of 4-D AABB (axis-aligned bounding box) can be added to a 4-D body object for quick estimation of object intersections. The vertex class is defined as a subclass of 4-D vector class and just keeps four numbers for 4-D coordinate values. The authors have a plan to apply the compact representation method proposed by Blandford, et al. [3] for 2-D and 3-D meshes to 4-D mesh models.

5. FOUR-DIMENSIONAL MESH WEAVER

The 4-D mesh generation method described in Section 4 has been implemented as a C++ program named 4-D Mesh Weaver on PC. This program is running as a file converter from a time series of 3-D voxel files to a 4-D mesh file as shown in Fig. 11. Four-dimensional geometric modeling is not a new idea itself, but it has been considered as an impractical idea due to its larger data size, longer processing time, complex data structure, and hard-to-understand operations. The increasing capability of computer hardware and software is gradually resolving these problems so as to make the idea of 4-D geometric modeling practical. The authors expect that 4-D geometric modeling can contribute to find new approaches to spatio-temporal problems.

An example of 4-D mesh generation by a conventional approach and 4-D Mesh Weaver is summarized in Fig. 11 and Tab. 2. The conventional approach consists of two steps: mesh generation with duplicated vertices and merging vertices. The number of unmerged vertices is eight times of the merged vertices. In this example, merging process made elimination of vertices excessively and it causes inappropriate elimination of tetrahedrons. The processing time by 4-D Mesh Weaver is almost one-third of the conventional approach. The 4-D Mesh Weaver can translate 3-D voxel files of 32 time steps into a 4-D mesh model. This means that three voxel data in 128^3 format can be processed in each second, using only single CPU. The target of 4-D Mesh Weaver is to process ten 3-D scenes in 1024_3 resolution in every second. The parallel processing using multiple CPU, GPGPU, and cloud computing environment will encourage this challenge.

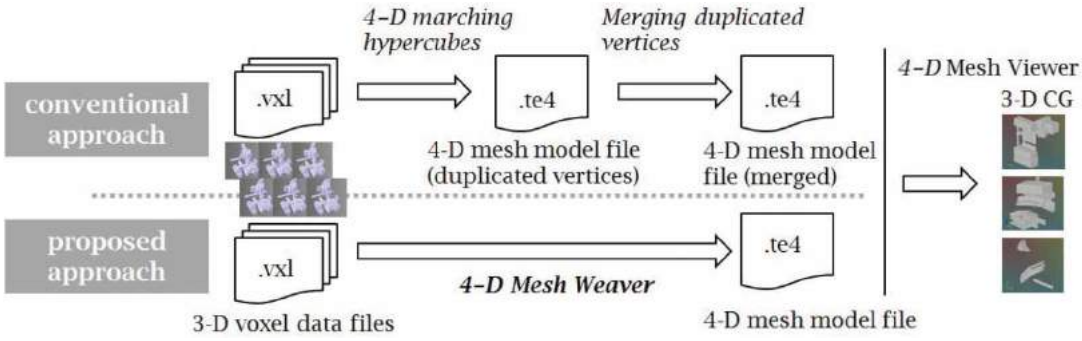

Fig. 11: Model data processing by 4-D Mesh Weaver.

Input Models	Voxel files X,Y,Z:128, T:32 value 0/1 2.05MB/file
PC specification	Windows 7, Intel Corei7 870 2.93GHz, 16GB, Visual Studio 2010 C++, 64bits

	Unmerged vertices	Merged vertices	Tetrahedrons	Process Time[sec]
Conventional	27,684,470	2,326,981	16,035,881	32.3
4-D Mesh Weaver	N/A	3,460,588	20,784,259	10.3

Generated 4-D mesh model file 379MB

Tab. 2: Comparison of 4-D mesh generation between conventional approach and 4-D Mesh Weaver.

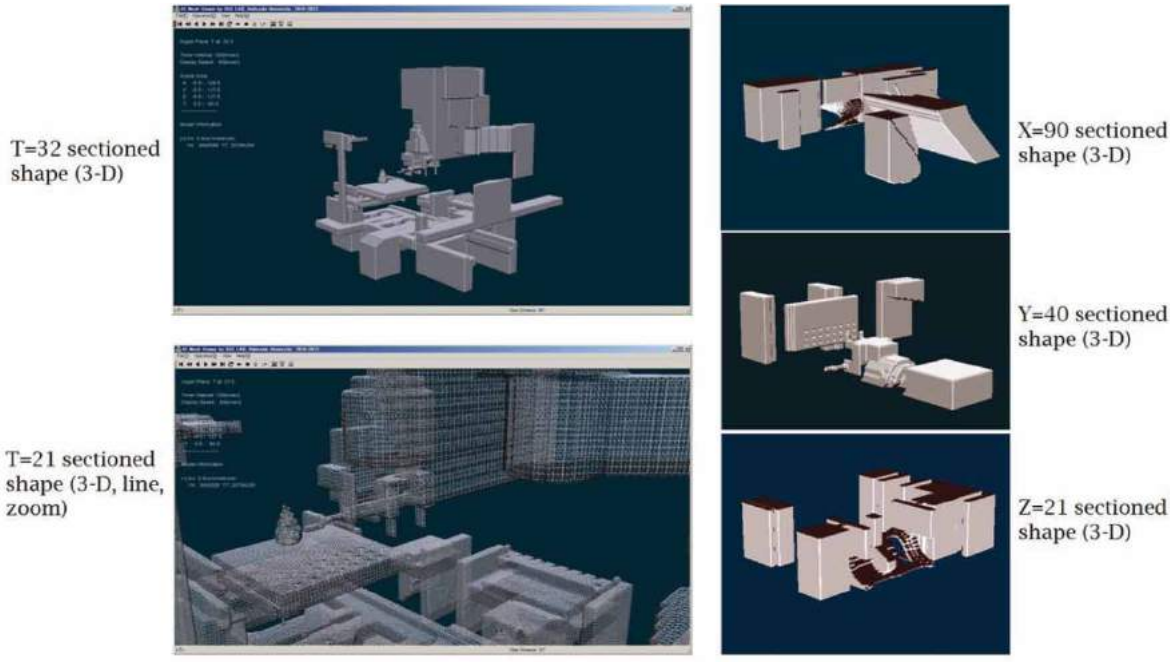

Fig. 12: Hyperplane cutting views of 4-D mesh model by 4-D Mesh Viewer.

Fig. 12 shows the cutting sections of the 4-D mesh model used in the example. Because we cannot see a 4-D mesh model directly, 4-D Mesh Viewer cuts a 4-D mesh model by a specified hyperplane, extracts cross-sections of tetrahedrons, and forms a 3-D mesh model for computer graphics. When we cut a 4-D mesh model with a series of time-orthogonal hyperplane step-by-step, we can reproduce the animation of dynamic objects.

6. CONCLUSIONS

Four-dimensional geometric modeling is not a new idea itself, but it has been considered as an impractical idea due to its larger data size, longer processing time, complex data structure, and hard-to-understand operations. The increasing capability of computer hardware and software is gradually resolving these problems so as to make the idea of 4-D geometric modeling practical.

This paper dealt with only a small part in the 4-D geometric modeling framework. Input and output data were limited to a series of 3-D voxel models and a 4-D mesh model respectively. This combination is the first step and the authors have a plan to examine various combinations of spatio-temporal input data models and 4-D modeling methods. For example, the authors are now developing a 4-D shape representation and operation method based on implicit function. The approach of 4-D point cloud and 4-D Delaunary triangulation by Aganj, et al.[1] will expand the possibility of 4-D geometric modeling.

The authors expect that 4-D geometric modeling can contribute to find new approaches to spatio-temporal problems in various fields such as design, manufacturing, robotics, computer vision, medicine, geoscience and so on.

ACKNOWLEDGEMENT

This work was partly supported by JSPS Grant-in-Aid for Challenging Exploratory Research, Grant Number 24656106.

REFERENCES

[1] Aganj, E.; Pons, J-P.; Keriven R.: Globally optimal spatio-temporal reconstruction from cluttered videos. Proc. 9th Asian Conference on Computer Vision, 2009, 667–678.

[2] Bhaniramka, P.; Wenger, R.; Crawfis, R.: Isosurfacing in higher dimensions, Proc. of the conference on Visualization '00, 2000, 267–273.

[3] Blandford, D.K.; Blelloch, G.E.; Cardoze, D.E.; Kadow, C.: Compact Representations of Simplicial Meshes in Two and Three Dimensions, International Journal of Computational Geometry & Applications, 15(1), 2005, 3–23.

[4] Brock, K.M., et al.: Automated generation of a four-dimensional model of the liver using warping and mutual information, Medical Physics, 30(6), 2003, 1128–1133. DOI:10.1118/1.1576781

[5] Cameron, S.: Collision Detection by Four-Dimensional Intersection Testing, IEEE Transactions on Robotics and Automation, 6(3), 1990, 291–302.

[6] Eisemann, E., et al.: Single-Pass GPU Solid Voxelization for Real-Time Application, Proc. Graphics Interface 2008, 2008, 73–80.

[7] Ji, G.; Shen, H.-W.; Wenger, R.: Volume Tracking Using Higher Dimensional Isosurfacing, 2003, Proc. the 14 th IEEE Visualization Conference, 209–216.

[8] Kameyama, H.; Otomo, I.; Onosato, M.; Tanaka, F.: Representing Continuous Process of Workpiece Transformation in Five-Axis Machining Using Spatio-Temporal Model, Proc. of 2012 Asian Conference on Design and Digital Engineering, 2012, #100098

[9] Lorensen, W.E.; Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm, Computer Graphics, 21(4), 1987, 163–169.

[10] Shen, H.-W.: Time-Dependent Isosurface Extraction, in Charles, D., et al. eds. The Visualization Handbook, Academic Press, 2011, 57–67.

[11] Wyvill, G.; McPheeters, C.; Wyvill, B.: Data structure for soft objects, The Visual Computer, 1986, 2, Springer-Verlag, 227–234.

[12] Zhang, J.P.; Hu, Z.Z.: BIM- and 4D-based integrated solution of analysis and management for conflicts and structural safety problems during construction: 1. Principles and methodologies, Automation in Construction, 20, 2011, 155–160. DOI:10.1016/j.autcon.2010.09.013.