LG-006

ボクセル集合からの四面体モデル生成の線形計算量アルゴリズム

Voxel Cloud Tetrahedralization with Linear Complexity

チウン チュンズン† TrungDung Truong　北坂 孝幸† Takayuki Kitasaka　森 健策† Kensaku Mori　末永 康仁† Yasuhito Suenaga

1．はじめに

四面体を用いた三次元物体の形状表現は，有限要素法（FEM）による構造解析を始めとする多くの工学分野で応用されてきた[1][2]．特に，近年の医用イメージング技術の発展により人体内の臓器を三次元ボクセル集合として取得可能となったため，生体組織の形状表現に関する研究が盛んに行われるようになった [3]．また，その応用例として，仮想的な手術環境を提供する手術シミュレータというものが挙げられる[4][5][6][7].

現在のシミュレータは，手術時の手順と感覚を把握するための訓練を目的とし，組織形状の正確さよりも術具と接触したときの動きの一貫性を重要視する．そのため，手術に関係のある組織の大まかな形状を予め構築しておき，術具と接触したときの形状変形を FEM で正確に計算してその様子を描画するのが主流である．しかし，この方法では実患者に対応させるのは難しい．

一方，筆者らは患者自身の CT 像を用いてその患者固有の解剖学的情報を形状表現に正確に反映できる全く新種の手術シミュレータの実現を目指している．本シミュレータでは，CT 像から組織領域を自動抽出し，その形状を表す四面体モデルも自動生成する．シミュレーション時においては動きの一貫性だけでなく組織変形の正確さを重視し，必要であれば形状モデルの再構築も可能とする．そのためには四面体モデルの高速生成が重要となる．

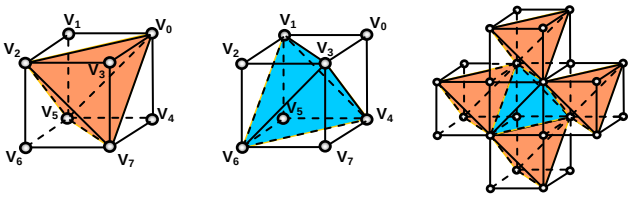
本稿では，上記シミュレータを実現するための基盤技術として，任意のボクセル集合から四面体形状を線形計算量で自動生成できる手法を検討したので報告する．多面体を線形計算量で分割する手法はあるが[2]，穴を有する形状には適用できない．穴に対処可能かつ計算量が O(*n*log*n*)と予想される手法[3]は，2 つの等値面間の領域を分割するが，対話操作による等値面の選択が必要である．

以下，2 で実現したい四面体モデルを述べ，3 でそれを記述するためのデータ構造を示す．4 で四面体モデルの生成アルゴリズムを説明する．5 で人工的に生成したボクセル集合および，実人体三次元 CT 像から抽出した臓器領域に適用した実験結果を示し，考察を加える．

2．四面体モデル

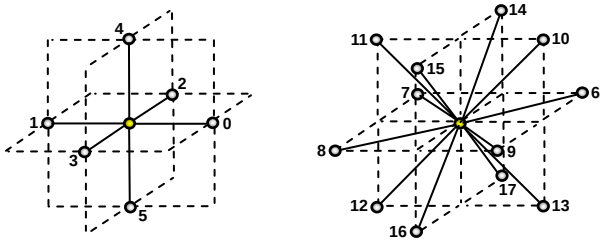
本稿で提案する四面体モデルはボクセル集合を囲む四面体から構成される．四面体は正四面体と直角四面体の 2 種類とし，正四面体 1 個と直角四面体 4 個が 1 個の六面体となるように配置する（図 1）．四面体の各頂点には節点を配置する．隣接する四面体は必ず 1 面（3 節点と 3 辺）を共有する．従って，同一面を共有する四面体の最大数は 2 であり，同一節点を共有する辺の最大数は 18 である（図 2）．本稿では，ばねモデルを用いた表現への応用 [8]を想定し，四面体間の接続は節点と辺だけで実現する．

†名古屋大学　大学院情報科学研究科
〒464－8603 愛知県名古屋市千種区不老町
Tel: 052-789-5699 Fax: 052-789-3815
E-mail: ttdung@suenaga.m.is.nagoya-u.ac.jp



(a) type A　(b) type B　(c) example

Fig. 1: Tetrahedral model



(a) 6-neighborhood　(b) 12-neighborhood

Fig. 2: Relation between a node and its neighbors

本四面体モデルは六面体の分割により生成できる．特に，座標軸と平行な六面体集合が作成できるので，三次元画像から抽出されたボクセル集合を用いて生成する場合では都合がよい．本稿では，ボクセル集合を一定間隔 d の格子で分割し，升目（セル）を要素とする参照テーブル（LUT：Look-up Table）を用いることでモデルの高速生成を図る．

3．データ構造

本稿では四面体モデルを四面体およびその節点と辺の集合で表す．節点（node）は，識別番号 *in*，座標値 x, y, z，およびそれを端点とする辺の識別番号 *eid*[18] を保有する．辺（edge）は識別番号 *ie*，種類 *etype*（0=辺候補, 1=辺），および端点節点の識別番号 *enid*[2] を保有する．四面体（tetrahedron）は識別番号 *it* と頂点節点の識別番号 *tnid*[4] を保有する．また，モデル生成にのみ必要なデータ構造として，六面体を用意する．六面体 （hexahedron）は識別番号 *ih*，LUT 参照用のインデックス i_x, i_y, i_z，頂点節点の識別番号 *hnid*[8]，および四面体の配置パターン *htype*（0=A, 1=B）を保有する．

4．四面体モデルの生成アルゴリズム

ここでは，ボクセル集合から均一サイズの六面体集合を生成し，それらを分割することで四面体モデルを得る方法について説明する．本モデルの生成処理は，前処理→六面体の生成→六面体の分割という流れで行う．

4.1　前処理

格子による空間分割：ボクセル集合上に一定間隔 d ボクセル（$d \geq 1$）の格子を配置し，格子点を中心としたセル集合を表す参照テーブル T_h を構築する（図 3）．まず，ボクセル集合 S_v に含まれるすべてのボクセル v の座標を調べて S_v のバウンディングボリューム（BV）を表す座標 x_v^{min}，x_v^{max}，y_v^{min}，y_v^{max}，z_v^{min}，z_v^{max} を求める．次に，これら

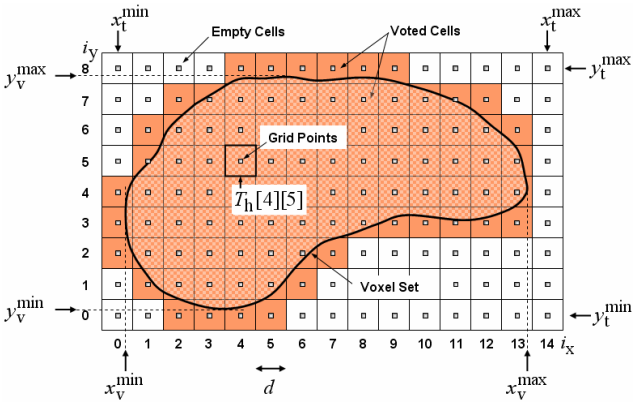


Fig. 3: Illustration of space partitioning and LUT (XY plane)

の座標を d で割り切れるように調節して新しい BV を表す座標 x_T^{min}，x_T^{max}，y_T^{min}，y_T^{max}，z_T^{min}，z_T^{max} を計算する（アルゴリズム 1）．この BV の各方向を d で分割し，格子の各点をセルの中心とする．最後に，式(1)で表される x, y, z 方向のセル数を格納できる配列を構築し，T_h とする．

$$\begin{aligned} N_x &= \lfloor (x_T^{max} - x_T^{min}) / d \rfloor + 1 \\ N_y &= \lfloor (y_T^{max} - y_T^{min}) / d \rfloor + 1 \\ N_z &= \lfloor (z_T^{max} - z_T^{min}) / d \rfloor + 1 \end{aligned} \quad (1)$$

ある点の座標 x, y, z が与えられたときそれを含むセル (i_x, i_y, i_z) は式(2)のように特定する．

$$\begin{aligned} i_x &= \lfloor (x + d/2 - x_T^{min}) / d \rfloor \\ i_y &= \lfloor (y + d/2 - y_T^{min}) / d \rfloor \\ i_z &= \lfloor (z + d/2 - z_T^{min}) / d \rfloor \end{aligned} \quad (2)$$

```
01 x_T^min = x_v^min ; x_T^max = x_v^max ;
02 while ( x_T^min mod d ≠ 0 ) do x_T^min = x_T^min − 1 ;
03 while ( x_T^max mod d ≠ 0 ) do x_T^max = x_T^max + 1 ;
```

Algorithm 1: Calculation of grid points bounding volume

ボクセル含有セルの決定：すべてのセルからボクセル 1 個以上を含むものを探す．具体的には，各々のボクセル（座標 x, y, z）についてそれに対応するセルを式(2)より算出し，そのセルに投票する．1 票以上を獲得したセルはボクセル含有セルとする．

4.2 六面体の生成

節点と六面体の生成：ボクセル含有セルに対して六面体を生成し，頂点に節点を生成する（アルゴリズム 2）．六面体 1 個に付き必ず 8 個の節点を生成する．そのため，節点数は六面体数×8 であり，同一座標をもつ節点は最大 8 点存在している．六面体と節点はバッファ B_h と B_n に格納する．B_h の要素数はボクセル含有セルの数とし，B_n の要素数はその 8 倍とする．

```
GenerateHexahedra(Input: T_h , d , B_h , B_n ; Output: N_h , N_n )
Constants: sx = {1,−1,−1, 1, 1,−1,−1, 1};
           sy = {1, 1,−1,−1, 1, 1,−1,−1};
           sz = {1, 1, 1, 1,−1,−1,−1,−1};
Variables: i_h , i_n , i_x , i_y , i_z
01 i_h = 0 ; i_n = 0 ;
02 for each cell (i_x , i_y , i_z ) do
03   begin
04     if ( T_h[i_x][i_y][i_z] > 0 ) then
05       begin
           // Generate new hexahedron
06         B_h[i_h].ih = i_h ; T_h[i_x][i_y][i_z] = i_h ;
07         B_h[i_h].i_x = i_x ; B_h[i_h].i_y = i_y ; B_h[i_h].i_z = i_z ;
           // Generate eight nodes at eight vertices
08         for each j ∈ [0,7] do
09           begin
10             B_n[i_n].x = x_T^min + i_x × d + sx[ j ]× d / 2 ;
11             B_n[i_n].y = y_T^min + i_y × d + sy[ j ]× d / 2 ;
12             B_n[i_n].z = z_T^min + i_z × d + sz[ j ]× d / 2 ;
13             B_h[i_h].hnid[ j ] = i_n ; B_n[i_n].in = i_n ;
14             i_n = i_n + 1 ;
15           end
16         i_h = i_h + 1 ;
17       end
18     else T_h[i_x][i_y][i_z] = −1 ;
19   end
20 N_h = i_h ; N_n = i_n ;
```

Algorithm 2: Generation of nodes and hexahedra

不要な節点のマーキング：同一座標をもつ節点を後で削除できるようにマークする．まず，生成された節点数と同じ要素数のマーキング用バッファ B_{mark} を用意し，全要素を 0 で初期化する．次に，最初に生成された六面体を除いたすべての六面体 h に対して h の節点（識別番号 i_n）と同一座標の節点（最大 7 点）を h の近傍六面体（最大 7 個）から取得する．最後に，取得した節点のうち識別番号が最も小さいものを節点 i_n の代わりとして h の頂点に配置すると同時に i_n をマークする（$B_{mark}[i_n] = 1$ とする）．

不要な節点の削除：マークされていない節点に新たな番号を割り当て，六面体に再配置する（アルゴリズム 3）．

```
RemoveUnnecessaryNodes(Input: B_h , B_n , B_mark ; Output: N_n )
Variables: i_n , i_n^*
01 i_n^* = 0 ;
02 for each node p ∈ B_n do
03   if ( B_mark[ p.in ] == 0 ) then
04     begin
05       B_mark[ p.in ] = i_n^* ; B_n[i_n^*] = p ; B_n[i_n^*].in = i_n^* ;
06       i_n^* = i_n^* + 1 ;
07     end
08 N_n = i_n^* ;
09 for each hexahedron h ∈ B_h do
10   for each j ∈ [0,7] do
11     begin
12       i_n = h.hnid[ j ] ; h.hnid[ j ] = B_mark[i_n] ;
13     end
```

Algorithm 3: Unnecessary nodes removal

4.3 六面体の分割

節点の関連付け：各々の六面体に対してその頂点にある節点同士の関係を設定する．ある節点 p と近傍節点 q との関連付けは，それらの識別番号を互いの配列 eid に格納することで実現できる．ここでは，関連付けられた節点から辺を生成する際の効率化を図るために，配列 $p.eid$ の全要素を 1 で初期化し，q の識別番号に 1 を足した結果を符合反転させて $p.eid$ に格納する．例えば六面体 h の頂点 V_0 と V_1 および V_0 と V_2 の関連付けはアルゴリズム 4 のようになる（図 1，2 も参照されたい）．

```
01 B_n[ h.hnid[0] ].eid[1] = −(h.hnid[1]+1);
02 B_n[ h.hnid[1] ].eid[0] = −(h.hnid[0]+1);
03 B_n[ h.hnid[0] ].eid[8] = −(h.hnid[2]+1);
04 B_n[ h.hnid[2] ].eid[6] = −(h.hnid[0]+1);
```

Algorithm 4: Node connection

辺の生成：関連付けられた節点を線分で結び，四面体の辺と成り得るものとならないものに分類する．まず，生成さ

れる線分を格納するためのバッファ B_e を用意する．次に，各々の節点に対してそれと関連付けられた節点とを結ぶ線分（辺候補）を生成し，その識別番号を B_e と両節点の *eid* に格納する（アルゴリズム 5）．次に，辺候補の中から六面体を四面体に分割する際に四面体の辺と成り得るものだけを辺とする（アルゴリズム 6）．具体的には，任意の六面体 h に対して四面体の配置パターン（図 1(a)又は(b)）を設定し，それに従って不要な辺候補（例えば，配置パターン A では頂点 V_3 と V_4 を結ぶ辺候補）を除外する．h に対する処理を終えたら h に隣接する六面体に対して h と異なった配置パターンを設定し，h のときと同様な処理を行う．

```
GenerateEdgeCandidates(Input: B_n ; Output: B_e , N_e )
Variables: i_e , j , k ;
Constants: qi = {1, 0, 3, 2, 5, 4, 8, 9, 6, 7, 12, 13, 10, 11, 16, 17, 14, 15};
01 i_e = 0 ;
02 for each node p ∈ B_n do
03 for each j ∈ [0, 17] do
04   begin
05     if ( p.eid [ j ] < 0 ) then
06       begin
07         q = B_p[ −( p.eid [ j ] + 1) ] ;
08         k = qi[ j ] ;
09         if ( q.eid [k ] < 0 ) then
10           begin
11             B_e[i_e ].ie = i_e ; p.eid [ j ] = q.eid [k ] = i_e ;
12             B_e[i_e ].enid [0] = p.in ;
13             B_e[i_e ].enid [1] = q.in ;
14             B_e[i_e ].etype = 1 ; i_e = i_e + 1 ;
15           end
16       end
17   end
17 N_e = i_e ;
```

Algorithm 5: Generation of edge candidates

```
DetermineTetrahedralEdges(Input: B_h , type )
01 for each hexahedron h ∈ B_h do h.htype = −1 ;
02 for each hexahedron h ∈ B_h do
03   InvalidateUnnecessaryCandidates ( h , type , B_e , B_n );
InvalidateUnnecessaryCandidates(Input: h , type , B_e , B_n )
Variables: i_e ;
01 if ( h.htype < 0 ) then
02   begin
03     if ( type == A ) then
04       begin
           // Invalidate the edge connecting vertices V_3 and V_4
05         i_e = B_n[ h.hnid [3] ].eid [17] ;
06         B_e[i_e ].etype = 0 ;
07         Do the same for V_3 and V_1 , V_3 and V_6 , and so forth;
08       end
09     else if ( type == B ) then
10       begin
           // Invalidate the edge connecting vertices V_0 and V_2
11         i_e = B_n[ h.hnid [0] ].eid [8] ;
12         B_e[i_e ].etype = 0 ;
13         Do the same for V_0 and V_5 , V_0 and V_7 , and so forth;
14       end
15     h.htype = type ;
16     for each hexahedron h' (one of six neighbors of h ) do
17       InvalidateUnnecessaryCandidates ( h' , 1 − type , B_e , B_n );
18   end
```

Algorithm 6: Edge determination

四面体の生成：節点と辺のデータより六面体を分割して四面体集合を生成する（アルゴリズム 7）．

```
GenerateTetrahedra (Input: B_h ; Output: B_t )
Variables: i_t , j , h ;
Constants: t_0^A = {0, 2, 3, 7};  t_0^B = {0, 1, 3, 4};
           t_1^A = {0, 4, 5, 7};  t_1^B = {2, 1, 6, 3};
           t_2^A = {0, 1, 2, 5};  t_2^B = {5, 1, 4, 6};
           t_3^A = {0, 5, 6, 7};  t_3^B = {7, 3, 6, 4};
           t_4^A = {0, 5, 2, 7};  t_4^B = {1, 3, 4, 6};
01 i_t = 0 ;
02 for each hexahedron h ∈ B_h do
03   if ( h.htype == A ) then
04     begin
         // Generate the first of five tetrahedra divided from h
05       B_t[i_t ].it = i_t ;
06       for each j ∈ [0, 3] do
07         B_t[i_t ].tnid [ j ] = h.hnid [ t_0^A [ j ] ] ;
08       i_t = i_t + 1 ;
09       Do the same for the second, the third, and so forth;
10     end
11   else if ( h.htype == B ) then
12     begin
         // Generate the first of five tetrahedra divided from h
13       B_t[i_t ].it = i_t ;
14       for each j ∈ [0, 3] do
15         B_t[i_t ].tnid [ j ] = h.hnid [ t_0^B [ j ] ] ;
16       i_t = i_t + 1 ;
17       Do the same for the second, the third, and so forth;
18     end
```

Algorithm 7: Generation of tetrahedra

4.4 アルゴリズムの特徴

本稿で提案したアルゴリズムは任意のボクセル集合に適用可能である．ただし，空洞を有する集合や，連結しない複数の集合などは，格子間隔の大きさによっては空洞や非連結成分が再現されない可能性がある．また，入力にノイズなどが含まれた場合，1 辺あるいは 1 節点しか共有しない四面体が生成される可能性もある．

アルゴリズムの計算量はボクセル数 N_v のオーダーである．時間計算量の内訳は，前処理が $O(N_v)$，節点と六面体の生成が $O(N_x \times N_y \times N_z)$，不要節点のマーキングが $O(N_h)$，不要節点の削除が $O(N_n)$，節点の関連付けが $O(N_h)$，辺候補の生成が $O(N_n)$，辺の決定が $O(N_h)$，四面体の生成が $O(N_h)$ である．格子間隔を $d = 1$ とした場合では節点数 N_n がボクセル数 N_v を上回るが，それでも N_v の高々倍数であるため，全体の計算量を $O(N_v)$ と見なすことができる．一方，領域計算量は LUT とバッファ B_{mark} の要素数 $N_x \times N_y \times N_z$ と $N_h \times 8$ のオーダーであり，$O(N_v)$ と見なすことができる．

5. 実験結果と考察

本稿で提案したアルゴリズムを人工的に生成した 8 つのボクセル集合と，実人体 3 次元 CT 像から抽出した 5 つの臓器領域に適用し，処理時間および使用メモリ量を計測した．人工図形は球形とし，半径を 8 から 64 まで変化させて作成した．一方，実人体 CT 像から抽出したのは胃，大腸，肝臓，肺，気管支の領域である．なお，計算機は CPU：Xeon 3.6 GHz，Memory：2 GB，OS：Windows XP の PC を使用した．生成された四面体モデルデータおよび測定結果を表 1 に，モデルの 3 次元表示を図 4 に示す．

人工球での測定結果より，球の半径が 48 以下または格子

Table 1: Generated model data, processing time [sec], and used memory [megabytes]

Voxel Set	N_v	d	Number of Cells N_c	N_h ($/ N_c \times 100$)	N_n	N_e	Pre-processing	Hexahedra Generation	Tetrahedra Generation	Total Time	Used Memory
Sphere r= 8	2,552	1	4,913	2,552 (52)	3,279	17,388	0.000 (0)	0.111 (64)	0.063 (36)	0.174	1.81
Sphere r=16	18,852	1	35,937	18,852 (52)	21,534	120,957	0.000 (0)	0.196 (45)	0.244 (55)	0.440	12.50
Sphere r=32	143,760	1	274,625	143,760 (52)	153,895	1,908,831	0.047 (2)	0.687 (27)	1.828 (71)	2.562	123.02
Sphere r=48	477,920	1	912,673	477,920 (52)	500,391	6,301,967	0.203 (3)	2.125 (27)	5.452 (70)	7.780	404.98
Sphere r=56	755,476	1	1,442,897	755,476 (52)	785,903	6,919,971	0.328 (0)	5.565 (1)	495.501 (99)	501.394	546.14
Sphere r=64	1,123,772	2	274,625	145,426 (53)	155,629	1,930,761	0.140 (6)	0.656 (26)	1.703 (68)	2.499	139.32
Sphere r=64	1,123,772	4	35,937	19,435 (54)	22,136	124,512	0.109 (13)	0.344 (41)	0.391 (46)	0.844	29.71
Sphere r=64	1,123,772	8	4,913	2,801 (57)	3,564	18,990	0.109 (37)	0.093 (32)	0.092 (31)	0.294	19.07
Stomach	563,348	4	181,440	47,876 (26)	54,463	306,599	0.271 (12)	0.810 (36)	1.191 (52)	2.272	40.58
Colon	5,085,660	6	230,608	31,779 (14)	40,946	217,443	0.485 (22)	0.609 (28)	1.095 (50)	2.189	101.65
Liver	2,579,972	4	193,830	45,468 (23)	52,446	293,416	0.250 (11)	0.750 (34)	1.188 (54)	2.188	70.23
Lung	8,920,253	4	423,035	153,113 (36)	171,921	1,452,169	0.888 (22)	1.433 (35)	1.744 (43)	4.065	247.24
Bronchus	199,541	4	150,144	9,733 (6)	14,859	72,807	0.047 (5)	0.297 (31)	0.625 (64)	0.969	12.06

※Numbers inside round brackets indicate percentage

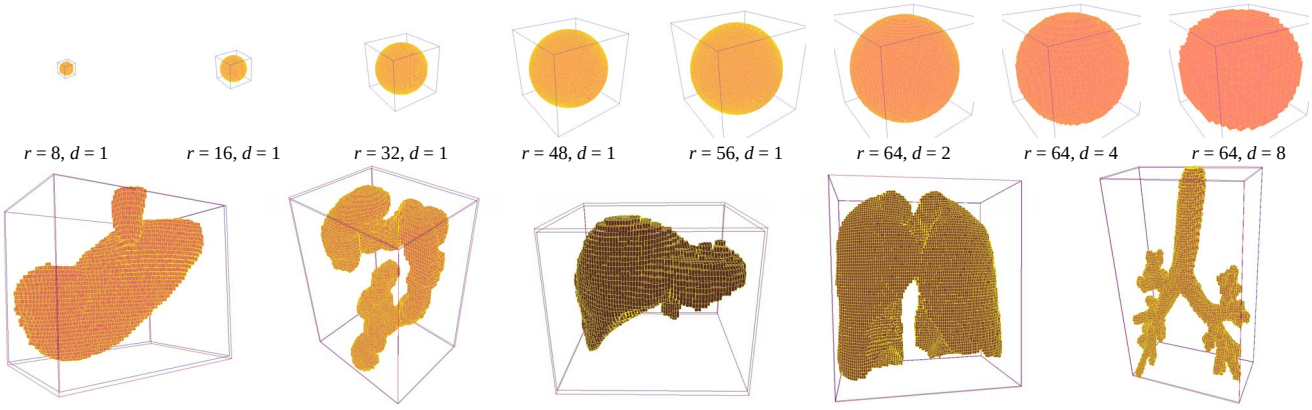


Fig. 4 Rendered images of tetrahedral models generated from synthetic voxel sets (upper row) and human CT images (lower row)

間隔が $d > 2$ のとき，処理時間と使用メモリ量はボクセル数と格子間隔の変化と線形関係にあることが分かる．しかし，$d = 1$ で半径を 56 以上にすると辺の数が膨大になるため，連続したメモリが確保できなくなり，プログラムが不正終了することがあった．そこで，半径が 56 のときだけ少量のメモリを複数回に分けて確保するようにアルゴリズムを変更した．r = 56 で処理時間が急増したのはそのためである．また，生成されるセル数が同じであっても（例えば r = 8, d = 1 と r = 64, d = 8）入力ボクセル数が異なるため，生成されるモデルの要素数や使用メモリ量も若干異なっている．

実人体 CT 像を用いた実験結果より，本アルゴリズムは穴を有す構造（胃，大腸，気管支），キノコ構造（肝臓），洞窟構造（肺），分岐構造（気管支）に適用できることが分かる．これらの臓器領域はバウンディングボリュームの 40%以下しか占めていないため，格子間隔が小さい場合では LUT の構築だけでも膨大なメモリが必要となり，本アルゴリズムが効果的に機能しなくなると考えられる．特に，手術シミュレータでは複数の臓器が複雑に絡み合ってもそれらの形状を同時に生成する必要があるため，メモリを効率的に利用できるアルゴリズムを今後検討する必要がある．

6. むすび

本稿では，患者個人の解剖学的情報が反映可能な手術シミュレータを実現するための基盤技術として，任意のボクセル集合から四面体形状を線形計算量で自動生成できるアルゴリズムを提案した．本アルゴリズムを人工的に生成したボクセル集合および，実人体三次元 CT 像から抽出した臓器領域に適用した結果，格子間隔を制限した上で本アルゴリズムが有効であることが分かった．今後の課題として，バウンディングボリューム内の領域割合を効果的に利用できる空間分割法の考案，臓器の切開および変形シミュレーションへの適用などが挙げられる．

謝辞 本研究の一部は，厚生労働省がん研究助成金，文部科学省・日本学術振興会科学研究費補助金，栢森情報科学振興財団助成金，および堀情報科学振興財団の研究助成金によった．

参考文献

[1] R. Schneiders, "A grid-based algorithm for the generation of hexahedral element meshes," Engineering with Computers, Vol.12, No.3-4, pp.168-177, 1996.
[2] D. Eppstein, "Linear complexity hexahedral mesh generation," Computational Geometry, Vol. 12, No. 1-2, pp.3-16, 1999.
[3] Y. Zhang et al., "3D finite element meshing from imaging data," Special issue of Computer Methods in Applied Mechanics and Engineering (CMAME) on Unstructured Mesh Generation, Vol. 194, No. 48-49, pp.5083-5106, 2005.
[4] S. Cotin et al., "A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation," Visual Computer, Vol. 16, No. 8, pp.437-452, 2000.
[5] 井上 悠介 他, "有限要素法及び可変形ボリュームレンダリングを用いた高画質な手術シミュレーションシステムの開発," 信学論, Vol.J87-D-II, No.1, pp.271-280, 2004.
[6] Surgical Science 社のホームページ, http://www.surgical-science.com/
[7] Simbionix 社のホームページ, http://www.simbionix.com/
[8] T.D. Truong et al, "安定した画像変形に基づく管腔臓器の仮想展開像生成法の改善," 信学論, Vol.J90-D, No.1, pp.138-151, 2007.