一种改进的 MC 算法

朱经纬 蒙培生 王 乘
(华中科技大学力学系工程计算与仿真研究所,武汉 430074)

摘 要 为了对等值面与子等值面进行提取和分组,在 MC 算法原理的基础上,提出了一种改进的等值面提取与子等值面分组算法。该算法首先将数据场分解为点、棱边、面与体元的拓扑结构;然后在整个数据场范围内求所有棱边与等值面的交点,并在面内连接交点形成面与等值面的交线,交线在体元内连接生成空间多边形;接着通过三角化各个体元内的空间多边形得到由顶点表与三角形表组成的等值面数据;最后根据三角形在顶点处的连接关系,采用种子算法对属于同一子等值面的三角形与顶点进行标记,属于同一子等值面的顶点与三角形将被存放在独立的顶点表与三角形表中。实验结果表明,该算法可以高效地实现等值面提取与子等值面的分组。

关键词 Marching Cubes 算法 子等值面 种子算法
中图法分类号:TP391.41 **文献标识码**:A **文章编号**:1006-8961(2008)07-1359-08

An Improved Marching Cubes Algorithm

ZHU Jing-wei, MENG Pei-sheng, WANG Cheng
(*Department of Mechanics, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract Based on the theory of Marching Cubes algorithm, an improved algorithm to extract isosurface from the 3D data field was introduced. At first the 3D data field was decomposed to the topological structure of points, lines, aces and cubes. Intersection points of the lines with the isosurface were calculated at first in the 3D data field. The intersection lines of the faces with the isosurface were gained by joining the intersection points in the faces. Intersection lines were joined together to form the space polygons in the cubes. The triangle mesh of the isosurface would be obtained by making triangle of the space polygons in each cube; The triangle mesh would be registered by a vertex table and a triangle table. Based on the relationship of connection among triangles at the vertexes, the seed algorithm was used to mark the vertexes and the triangles belong to the same child isosurface. And then vertexes and triangles belonging to the same child isosurface were registered by the individual vertex table and triangle table. It was proved by the instance that the algorithm could extract isosurface and group child isosurfaces with high efficiency.

Keywords Marching Cubes algorithm (MC), child isosurface, seed algorithm

1 引 言

MC 算法是 Lorensen 等人于 1987 年提出的一种 3 维重建算法[1,2]。MC 算法是面显示算法的一种,它根据阈值在一个 3 维数据场中抽取等值面实现 3 维重建。自从 MC 算法提出以来,通过不断改进,已广泛地应用在可视化的各个领域[3~12]。MC 算法提取的等值面是散乱的三角形集合,不能有效地区分子等值面,也不能反映三角形之间的连接关系,对于后续的交互操作或网格简化带来很大的不便。本文根据数据场的拓扑结构在数据场内定义棱边、面、体元等几何元素,首先判断并计算棱边与等值面的交点,在面内连接交点连线

基金项目:国防预研基金项目
收稿日期:2006-09-13;**改回日期**:2007-03-26
第一作者简介:朱经纬(1975~),男。华中科技大学工程力学专业博士研究生。主要研究方向为计算机图形学、3 维可视化。E-mail: zhu_jingwei@163.com

形成面与等值面的交线,交线在体元内连接生成空间多边形。三角化各个体元内的空间多边形得到由顶点表与三角形表组成的等值面数据。采用种子算法对属于同一子等值面的三角形与顶点进行标记,将属于同一子等值面的顶点与三角形提取出来存放在独立的顶点表与三角形表中。实验结果表明,本文的算法可以快速提取等值面,并有效地实现子等值面的分组。

2 MC 算法

3 维数据场的数据点分布在网格节点上。相邻的 8 个数据点组成的立方体称为体元。如图 1 所示,设定阈值 C,如果对于某个体元,有部分顶点数据值大于 C,而另一些顶点数据值小于等于 C,则值为 C 的等值面与该体元相交。MC 算法在数据场内逐一寻找与等值面相交的体元,如果等值面与体元相交,通过插值计算体元棱边与等值面的交点,交点之间的连线近似地表示等值面与体元面的交线,交线在体元内首尾相连形成空间多边形,近似表示穿越体元内部的等值面,通过三角化空间多边形得到等值面三角形。当对所有的体元操作完成,得到的三角形集合就是数据场内的等值面。MC 算法可以有效地由数据场内抽取等值面,但是 MC 算法有以下的缺点,因为 MC 算法逐一处理体元,在数据场内如果棱边与等值面相交,该棱边被 4 个体元包含,棱边与等值面的交点将会在 4 个体元内计算 4 次,存在重复计算问题。MC 算法提取的等值面是散乱的三角形集合,不能区分子等值面,也不能反映三角形相互之间的连接关系。

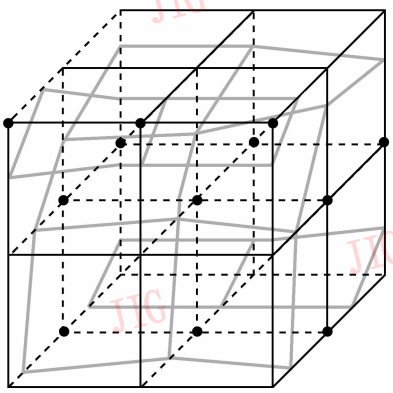

图 1 空间数据场内的等值面

Fig. 1 Isosurface in 3D data field

3 改进的 MC 算法设计

本文采用改进的策略提取等值面。将数据场分解为点、棱边、面与体元的拓扑结构。逐一检测棱边,如果棱边与等值面相交,计算出交点,将交点记入顶点表内。如果某棱边与等值面相交,包含该棱边的面与等值面必然相交,将交点的索引记入包含该棱边的面。如果面与等值面相交,包含该面的体元与等值面必然相交,根据面内记录的交点索引将交点相连,将连线加入包含该面的体元内,交线将相互连接形成体元内空间多边形。三角化各个体元内的空间多边形得到由顶点表与三角形表组成的等值面数据。三角形表内记录的顶点数据为顶点表内的顶点索引,顶点表的节点内记录包含该顶点的三角形,根据三角形在顶点处的连接关系,采用种子算法对属于同一子等值面的三角形与顶点进行标记,将属于同一子等值面的顶点与三角形提取出来存放在独立的顶点表与三角形表内。

3.1 空间数据场的拓扑结构与等值面的提取

如图 2(a)所示,3 维空间数据场内数据点分布在网格的节点上,节点记为 $P(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}$、$1 \leqslant j \leqslant j_{\max}$,$1 \leqslant k \leqslant k_{\max}$,相邻的 8 个数据点组成的立方体称为体元,该体元命名为 $C(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}-1$、$1 \leqslant j \leqslant j_{\max}-1$,$1 \leqslant k \leqslant k_{\max}-1$。如图 2(b)所示,相邻数据点的连线称为棱边,棱边可分为 3 类:沿 I 轴方向的棱边,记为 $LI(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}-1$、$1 \leqslant j \leqslant j_{\max}$,$1 \leqslant k \leqslant k_{\max}$;沿 J 轴方向的棱边,记为 $LJ(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}$、$1 \leqslant j \leqslant j_{\max}-1$,$1 \leqslant k \leqslant k_{\max}$;沿 K 轴方向的棱边,记为 $LK(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}$、$1 \leqslant j \leqslant j_{\max}$,$1 \leqslant k \leqslant k_{\max}-1$。如图 2(c)所示,相邻的 4 个数据点形成一个面,根据面与轴线的相对关系,面可分为 3 类:垂直 I 轴方向的面 $SI(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}$、$1 \leqslant j \leqslant j_{\max}-1$,$1 \leqslant k \leqslant k_{\max}-1$;垂直 J 轴方向的面 $SJ(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}-1$、$1 \leqslant j \leqslant j_{\max}$,$1 \leqslant k \leqslant k_{\max}-1$;垂直 K 轴方向的面 $SK(i,j,k)$,有 $1 \leqslant i \leqslant i_{\max}-1$、$1 \leqslant j \leqslant j_{\max}-1$,$1 \leqslant k \leqslant k_{\max}$。

根据设定的阈值 C,对于数据场中棱边,如果一个端点大于阈值 C,而另一个端点小于等于阈值 C,该棱边与等值面相交,棱边与等值面的交点可以通过两个端点坐标的线性插值求出。采用中心差分计算出端点的梯度,然后通过两个端点的梯度的线性

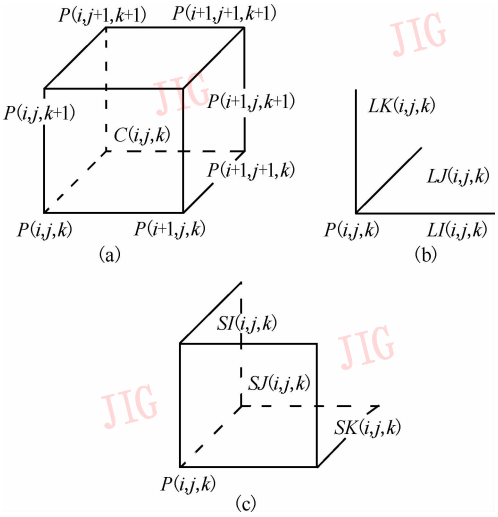

图 2　数据点与棱边,面和体元的相对关系

Fig. 2　Relationship among data points and lines, aces, cubes

插值求出交点的梯度就是该交点的法向量。棱边与等值面的交点,即为等值面三角化后三角形的顶点,将交点存放在顶点表 *V*_Temp 内,顶点表采用动态链表结构。链表的节点结构如下:

```
struct V      //顶点节点
{float  c[3];      //交点的 3 维坐标
float  n[3];      //交点的法向量
TL  *tl      //指向共有该顶点的三角形表
int      t_nub;      //共有该顶点的三角形的数目
int      indicate;      //子等值面标识
V      *next;      //指向下一个节点
}
struct TL      //共有顶点的三角形节点
{ T      *t;      //三角形节点指针
TL      *next;      //指向下一个节点
}
```

在顶点表内第 *n* 个节点记为 *V*_Temp(*n*)。一个顶点可能被多个三角形共有,在节点内有三角形链表,称为该顶点的三角形表,用以记录包含该顶点的三角形。

如果棱边与等值面相交,则包含该棱边的面与等值面也相交,面与等值面相交形成的交线可以用棱边与等值面交点的连线近似表示。如图 2 所示,边与包含该边的 4 个面的相对关系,给定某棱边,包含该棱边的面由表 1 给出。如图 3 所示,每个面包含 4 条棱边,在面内对棱边进行局部编号,如果给定某面,面包含 4 条棱边及其局部编号可由表 2 查得。根据面在空间的不同位置,采用 3 个面表记录面与等值面相交的情况:垂直 *I* 轴线方向的面表 *SI*_Tab、垂直 *J* 轴线方向的面表 *SJ*_Tab 和垂直 *K* 轴线方向的面表 *SK*_Tab。

表 1　棱边与面的相互关系表

Tab. 1　Relationship between line and ace

棱边	面 1	面 2	面 3	面 4
LI(*i*,*j*,*k*)	*SJ*(*i*,*j*,*k*−1)	*SJ*(*i*,*j*,*k*)	*SK*(*i*,*j*−1,*k*)	*SK*(*i*,*j*,*k*)
LJ(*i*,*j*,*k*)	*SI*(*i*,*j*,*k*−1)	*SI*(*i*,*j*,*k*)	*SK*(*i*−1,*j*,*k*)	*SK*(*i*,*j*,*k*)
LK(*i*,*j*,*k*)	*SI*(*i*,*j*−1,*k*)	*SI*(*i*,*j*,*k*)	*SJ*(*i*−1,*j*,*k*)	*SJ*(*i*,*j*,*k*)

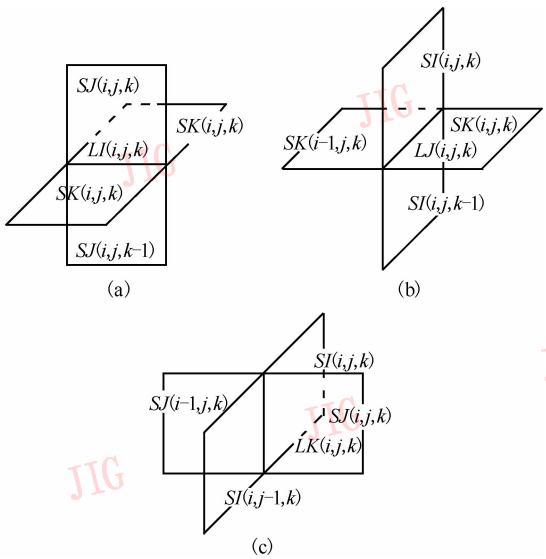

图 3　棱边与包含棱边的面

Fig. 3　Line and aces contain the line

表 2　面内各棱边的局部编号

Tab. 2　Local serial numbers of lines in ace

面	边 1	边 2	边 3	边 4
SI(*i*,*j*,*k*)	*LK*(*i*,*j*,*k*)	*LJ*(*i*,*j*,*k*)	*LK*(*i*,*j*+1,*k*)	*LJ*(*i*,*j*,*k*+1)
SJ(*i*,*j*,*k*)	*LK*(*i*,*j*,*k*)	*LI*(*i*,*j*,*k*)	*LK*(*i*+1,*j*,*k*)	*LI*(*i*,*j*,*k*+1)
SK(*i*,*j*,*k*)	*LJ*(*i*,*j*,*k*)	*LI*(*i*,*j*,*k*)	*LJ*(*i*+1,*j*,*k*)	*LI*(*i*,*j*+1,*k*)

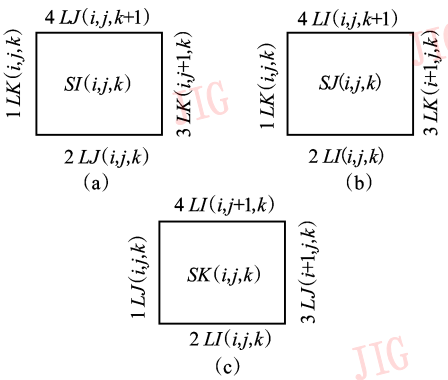

图 4　面内各棱边的局部编号

Fig. 4　Local serial number of lines in ace

上述的 3 个表均采用动态链表结构。链表的节点结构如下：

```
struct    S          // 面表节点
{int   IJK[3]        // 记录面的编号
V * v[4];            // 交点的指针
int   which[4];      // 交点所处棱边的局部编号
int   v_nub;         // 交点数目,初始值为 0
S     * next;        // 指向下一个节点
}
```

在 SI_Tab 中第 n 个节点记为 $SI\_Tab(n)$，面 $SI(i,j,k)$ 在 SI_Tab 中的节点内有 $IJK[0]=i$，$IJK[1]=j$，$IJK[2]=k$。

每一个面表有一个哈西表，分别为：SI_Tab 的哈西表 $S*SIH[pi]$，$pi=i_{max}*(j_{max}-1)*(k_{max}-1)$，$SI(i,j,k)$ 的键值

$$key=i_{max}*(j_{max}-1)*k+i_{max}*j+i \tag{1}$$

对应哈西表项为 $SIH[key]$，若 $SIH[key]!=NULL$，表明面 $SI(i,j,k)$ 的节点已经生成，$SIH[key]$ 指向 SI_Tab 内该节点，如果 $SIH[key]==NULL$，表明 $SI(i,j,k)$ 的节点未生成。SJ_Tab 的哈西表 $S*SJH[pj]$，$pj=(i_{max}-1)*j_{max}*(k_{max}-1)$，面 $SJ(i,j,k)$ 的键值

$$key=(i_{max}-1)*j_{max}*k+(i_{max}-1)*j+i \tag{2}$$

对应哈西表项为 $SJH[key]$；SK_Tab 的哈西表 $S*SKH[pk]$，$pk=(i_{max}-1)*(j_{max}-1)*k_{max}$，$SK(i,j,k)$ 的键值

$$key=(i_{max}-1)*(j_{max}-1)*k+(i_{max}-1)*j+i \tag{3}$$

对应哈西表项为 $SKH[key]$。

棱边检测算法如下：

(1)顺序检测沿 I 轴的各条棱边，若 $LI(i,j,k)$ 与等值面相交，顺序执行下列操作：

①计算出边 $LI(i,j,k)$ 与等值面的交点坐标以及交点的法向量，将新节点加入顶点表 V_Temp 内，顶点表长度 $h_1=h_1+1$，新节点为 $V\_Temp(h_1)$

②查表 1 可知包含该棱边的面为 $SJ(i,j,k)$、$SK(i,j-1,k)$、$SK(i,j,k)$ 和 $SJ(i,j,k-1)$；

③根据式(2)计算出面 $SJ(i,j,k)$ 的键值 key，若 $SJH(key)==NULL$，生成该面的节点加入面表 SJ_Tab 的尾部，面表长度 $h_2=h_2+1$，新节点为 $SJ\_Tab(h_2)$，在节点内 $IJK[0]=i$，$IJK[1]=j$，$IJK[2]=k$，哈西表项 $SJH[key]=\&SJ\_Tab(h_2)$；否则，该面的节点已经生成，$SJH[key]$ 指向该面在面表内的节点。在面的结点内，添加顶点信息 $V[v\_nub]=\&V\_Temp(h_1)$，$which[v\_nub]=2$，$v\_nub=v\_nub+1$；

④将交点加入面 $SJ(i,j,k-1)$ 在面表 SJ_Tab 中的节点；

⑤将交点加入面 $SK(i,j,k)$ 在面表 SK_Tab 中的节点；

⑥将交点加入面 $SK(i,j-1,k)$ 在面表 SK_Tab 中的节点；

(2)顺序检测沿 J 轴线棱边；

(3)顺序检测沿 K 轴线棱边；

如图 5 所示，面与包含该面的两个体元的相对关系，如果给定某面，包含该面的体元可由表 3 查得。如果该面与等值面相交，则两个相邻的体元与等值面也相交。面与等值面相交生成的交线用面的棱边与等值面交点的连线近似表示，如果存在两个交点，将两个交点相连即为面与等值面的交线；如果存在 4 个交点，交点的连接关系按照渐近线法[3]确定。体元的各个面与等值面的交线首尾相连自然形成空间多边形。建立体元表 C_Tab，采用动态链表结构。链表的节点结构如下：

```
struct   C          //体元表节点
{ CL * cl;          // 空间多边形表
  int    cl_nub;    // 空间多边形的数目,初始值 0
  C * next;         // 指向下一个节点
}
struct   CL         //空间多边形节点
{ V    *v[10];      // 空间多边形顶点
int   v_nub;        // 顶点的数目,初始值 0
int   be_joint;     //标志顶点列是否封闭,0 为非封闭,1 为封闭
CL    *next;        //指向下一个节点的指针
}
```

在体元表 C_Tab 中第 n 个节点记为 $C\_Tab(n)$。有体元哈西表 $C*CH[pc]$，其中 $pc=(i_{max}-1)*(j_{max}-1)*(k_{max}-1)$。$C(i,j,k)$ 的键值

$$key=(i_{max}-1)*(j_{max}-1)*k+(i_{max}-1)*j+i \tag{4}$$

表 3 面与体元的相互关系表

Tab. 3 Relationship between ace and cube

面	体元 1	体元 2
$SI(i,j,k)$	$C(i-1,j,k)$	$C(i,j,k)$
$SJ(i,j,k)$	$C(i,j-1,k)$	$C(i,j,k)$
$SK(i,j,k)$	$C(i,j,k-1)$	$C(i,j,k)$

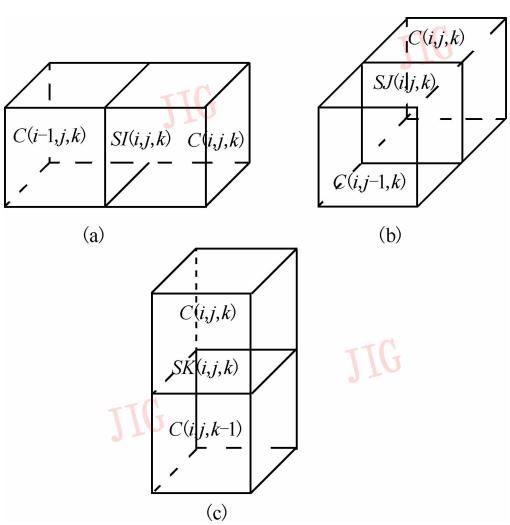

图 5　面与包含该面的体元

Fig. 5　Ace and cubes contain the line

对应哈西表项为 $CH[key]$,如果 $CH[key]$! = NULL,表明 $C(i,j,k)$ 的节点已经生成,$CH[key]$ 指向 C_Tab 内该节点,如果 $CH[key]$ = = NULL,表明 $C(i,j,k)$ 的节点未生成。

面表内信息加入体元表的算法如下:

(1)顺序处理表 SI_Tab 内的各个节点。节点 SI_Tab(n)内有 $i = IJK[0]$、$j = IJK[1]$,$k = IJK[2]$,表明节点记录的是面 $SI(i,j,k)$。查表 3 可知体元 $C(i-1,j,k)$ 与 $C(i,j,k)$ 包含该面。连接 $SI(i,j,k)$ 的棱边与等值面的交点得到 $SI(i,j,k)$ 与等值面的交线,并将交线记入 $C(i-1,j,k)$ 与 $C(i,j,k)$ 在体元表 C_Tab 中的节点,以 $C(i,j,k)$ 为例,执行以下步骤:

①根据式(4)计算体元 $C(i,j,k)$ 的哈西表键值 key,如果 $CH[key]$ = = NULL,生成该体元的节点加入体元表 C_Tab 的尾部,体元表长度 $h_1 = h_1 + 1$,新节点为 C_Tab(h_1),哈西表项 $CH[key]$ = &C_Tab(h_1);否则,该体元的节点已经生成,$CH[key]$ 指向体元表内该体元节点;

②将 SI_Tab(n)与等值面的交线逐条进行如下操作,交线的两个端点(V_1,V_2)与节点内纪录的多边形链表中未封闭的空间多边形顶点列逐条进行检测,并按照不同情况分别进行操作:V_1 与空间多边形顶点列中的头(尾)点重合,V_2 即不在该顶点列中,也不与其他的顶点列头(尾)点重合,将 V_2 加入顶点列的尾部;V_1 与空间多边形顶点列的头(尾)点重合,V_2 与该顶点列的尾(头)点重合,表明点列封闭;V_1 与空间多边形顶点列的头(尾)点重合,V_2 与其他的空间多边形顶点列的尾(头)点重合,将两个点列合并;如果上述情况均未出现,将 V_1 与 V_2 生成新的空间多边形点列加入节点内;

(2)顺序处理表 SJ_Tab 内的各个节点;

(3)顺序处理表 SK_Tab 内的各个节点;

完成上述步骤后,所有与等值面相交的体元存放在体元表之中,在每个节点中记录等值面与体元相交形成的空间多边形,将空间多边形三角化,生成等值面三角形。采用三角形表 T_Temp 记录等值面三角形,三角形表采用动态链表结构,链表的节点结构如下:

```
struct T          //三角形节点
{ V * v[3];       //顶点的指针
float  n[3];      //三角形的法向量
int    indicate;  //子等值面标识
T      * next;    //指向下一个节点
}
```

逐一处理体元表中节点,按照下述步骤对节点内记录的空间多边形进行三角化:

(1)体元表 C_Tab 中的节点 C_Tab(i)内记录的空间多边形数目为 cl_nub;$k = 1$;

(2)对于 C_Tab(i)内的第 k 个空间多边形,顶点数目为 $g = v$_nub;第 k 个空间多边形顶点列记为 $V(1),V(2),\cdots,V(g)$;

(3)$V(g)$、$V(g-1)$ 和 $V(g-2)$ 3 个顶点组成三角形,三角形的法向量为 3 个顶点法向量的平均,将新节点加入三角形表 T_Temp 内,三角形表长度 $h_1 = h_1 + 1$,新节点 T_Temp(h_1),T_Temp(h_1)内顶点指针指向顶点表的 3 个节点;在 $V(g)$、$V(g-1)$ 和 $V(g-2)$ 的三角形表内加入新三角形;$g = g - 3$;

(4)如果 $g > 0$,转步骤(3);否则转步骤(5);

(5)$k = k + 1$,如果 $k > cl$_nub,结束程序;否则转步骤(2);

经过上述操作,等值面三角形数据存放在三角形表 T_Temp 与顶点表 V_Temp 内。每个三角形表中的节点记录一个等值面三角形,节点内的顶点指针指向顶点表的 3 个节点;每个顶点表的节点纪录一个顶点,顶点的三角形表内记录共有该顶点的三角形。

3.2　子等值面数据分组

在上述的生成的等值面三角形数据中,所有 n

个顶点存放在一个顶点表中,所有 m 个三角形存放在一个三角形表中,并没有完成子等值面数据分组。顶点表与三角形表内各个节点的标志位均为 0,表示该节点没有经过标记。采用种子算法对属于同一子等值面的顶点和三角形进行相同的标记,通过标记,属于第 k 个子等值面的顶点和三角形的标记位为 k。种子算法需要两个种子表,分别是种子顶点表与种子三角形表。对三角形表 T_Temp 与顶点表 V_Temp 内的所有节点进行标记后,具有相同标记值的数据属于同一个子等值面,将属于同一个子等值面的顶点与三角形存放在独立的三角形表与顶点表内,均采用链表结构,三角形表节点采用 struct T,顶点表节点采用 struct V。第 h 个子等值面的顶点表记为 $V\_Tab(h)$,其中第 i 节点记为 $V\_Tab(h,i)$,顶点表内节点数目为 $V\_nub(h)$;三角形表记为 $T\_Tab(h)$,其中第 i 节点记为 $T\_Tab(h,i)$,三角形表内节点数目为 $T\_nub(h)$。

等值面数据标记与子等值面数据分组算法步骤:

(1) $k=0, h=0$;

(2)顺序遍历顶点表 V_Temp 的各个节点,如果检测到节点内子等值面标识满足 $indicate==0$,则 $k=k+1$, $indicate=k$,将该顶点加入种子顶点表,转步骤(3);如完成检索,子等值面数目为 k,转步骤(5);

(3)顺序检测种子顶点表内记录的各个顶点,如果包含该顶点的三角形的子等值面标识 $indicate==0$,将标识 $indicate=k$,并将该三角形加入种子三角形表。如果检测完种子顶点表内所有顶点后,种子三角形表为空,清空种子顶点表,转步骤(2);否则,清空种子顶点表,转步骤(4);

(4)顺序检测三角形种子表内记录的各个三角形,如果三角形的顶点的标识 $indicate==0$,则将标识 $indicate=k$,并将该顶点加入顶点种子表。如果检测完三角形种子表内所有三角形后,种子顶点表为空,则清空三角形种子表,转步骤(2);否则,清空三角形种子表,转步骤(3);

(5)如果 $h<k$, $h=h+1$,则生成第 h 子等值面的顶点表 $V\_Tab(h)$ 与三角形表 $T\_Tab(h)$, $V\_nub(h)=0$, $T\_nub(h)=0$,转步骤(6);否则,结束程序;

(6)顺序检索顶点表 V_Temp 的各个节点,若节点 $V\_Temp(i)$ 内 $indicate==h$,生成新节点加入顶点表 $V\_Tab(h)$ 尾部, $V\_nub(h)=V\_nub(h)+1$,由 $V\_Temp(i)$ 的三角形表找到包含该顶点的三角形在 T_Temp 内节点,将三角形节点中指向该顶点的指针更换为 $\&V\_Tab(h,V\_nub(h))$;所有顶点检索完毕,转步骤(7);

(7)顺序检索顶点表 T_Temp 的各个节点,如节点 $indicate==h$,生成新节点加入顶点表 $T\_Tab(h)$ 尾部, $T\_nub(h)=T\_nub(h)+1$,并根据 $T\_Tab(h,T\_nub(h))$ 内的 3 个顶点节点地址,将三角形节点的地址 $\&T\_Tab(h,T\_nub(h))$ 加入对应顶点的三角形表中;检索完毕,转步骤(5);

算法生成数据中的子等值面数据结构如图 6 所示,子等值面数据存放在独立的数据结构内,包括三角形表和顶点表,三角形表的每个节点纪录子等值面的一个三角形,节点内 3 个顶点指针分别指向顶点表的 3 个节点。顶点表的每个节点纪录子等值面的一个顶点,节点内纪录顶点坐标值,节点内还包含一个三角形表,三角形表内纪录了拥有该顶点的三角形节点的指针。

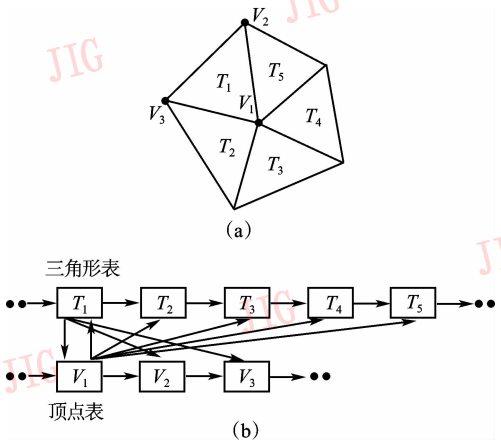

图 6 子等值面的数据结构

Fig. 6 Data structure of child isosurface

4 实验结果

采用传统 MC 算法以及改进 MC 算法对螺旋 CT 扫描数据进行 3 维重建,选用 pentium4 CPU 2.4G PC 机,内存 512M,采用 Visual C 6.0 编程。数据 A 规模为 68 层,每层为 512×512,层距 3mm,同层点距 0.4mm。数据 B 规模为 30 层,每层为 512×512,层距 3mm,同层点距 0.4mm。

对数据 A 采用传统 MC 算法进行 3 维重建,耗时 8.6s,生成的等值面共有 127 208 个三角形,需

要记录 127 208 ×3 个顶点数据,其中有很多顶点是重合的,有大量数据冗余。将三角形全部绘制如图 7(a)所示,传统 MC 算法逐一检测并处理体元,得到的等值面数据为散乱三角形集合,不能够区分子等值面,也不能反映三角形之间的连接关系。

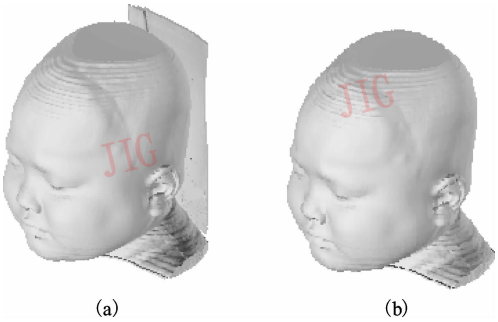

图 7　全体等值面与头部子等值面

Fig. 7　Whole isosurface and child isosurface of head

对数据 A 采用改进 MC 算法进行 3 维重建,提取全体等值面耗时 5.2s,全体等值面共有 127 208 个三角形,空间独立的顶点数目为 63 684。子等值面分组耗时 3.2s,等值面由 2 个子等值面组成,分别是头部子等值面有 101 766 个三角形,顶点数为 50 127;后部靠板子等值面 25 442 个三角形,顶点数为 13 557。头部子等值面如图 7(b)所示。

对数据 B 采用传统 MC 算法进行 3 维重建,耗时 4.9s,生成的等值面共有 67 208 个三角形,需要记录 67 208 × 3 个顶点数据,将三角形全部绘制如图 8(a)所示。

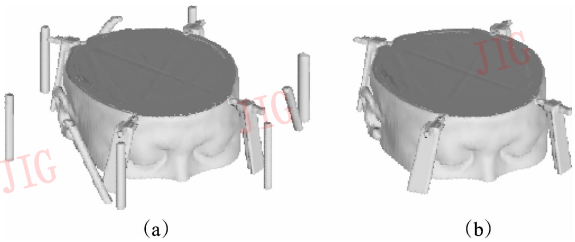

图 8　全体等值面与头部子等值面

Fig. 8　Whole isosurface and child isosurface of head

对数据 B 采用改进 MC 算法进行 3 维重建,提取全体等值面耗时 3.2s,全体等值面共有 67 208 个三角形,空间独立的顶点数目为 33 682。子等值面分组耗时 1.3s,等值面由 10 个子等值面组成,如图 8(b)所示,头部子等值面有 59 356 个三角形,顶点数为 29 714;可以看到,头部与支架相互连接,在等值面分组中,算法视其为一个子等值面;等值面内除头部子等值面,还有 9 个独立子等值面,9 个独立子等值面共有 7 852 个三角形,顶点数为 3 968。

5　结　论

MC 算法根据阈值在一个 3 维数据场中抽取等值面实现 3 维重建 ,算法逐一检测并处理与等值面相交的体元,顶点的计算过程中有大量重复计算,并且得到的等值面数据是散乱的三角形集合,数据不能有效地区分子等值面,也不能反映三角形的相互之间的连接关系。本文根据数据场的拓扑结构在数据场内定义棱边、面、体元等几何元素。数据场内棱边与等值面的交点即为等值面三角形的顶点,逐一判断并计算棱边与等值面的交点,空间独立的顶点只需要计算一次,并且独立的顶点在顶点表内有唯一的节点记录。算法提取的全体等值面数据用一个三角形表和一个顶点表记录。三角形表节点内的顶点数据为顶点表内的顶点索引,顶点表的节点内记录包含该顶点的三角形,从而可以明确等值面三角形之间的相互连接关系。采用种子算法对属于同一子等值面的三角形与顶点进行标记,将属于同一子等值面的顶点与三角形提取出来存放在独立的顶点表与三角形表内,实现子等值面数据的分组。

参考文献(References)

1　Lorensen W, Cline H. Marching Cubes: a high resolution 3D surface construction algorithm [J]. ACM Computer Graphics, 1987, **21**(4): 163 ~ 170.

2　Durst M J. Letters: Additional reference to "Marching Cubes" [J]. ACM Computer Graphics, 1988, **22**(4):72 ~ 73.

3　Nielson G M, Hamann B. The asymptotic decider: resolving the ambiguity in marching cubes [A]. In: Proceedings of IEEE Visualization[C], San Diego, CA, USA, 1992:83 ~ 91.

4　Skekhar R, Fayyad E, Yagel R. Octree-based decimation of marching cubes surfaces[A]. In: Proceedings of IEEE Visualization'96 [C], Los Alamitors, CA, USA, 1996:335 ~ 342.

5　Yuan Bin, Jin Qi-Jie. New method for resolving the ambiguity in marching Cubes[J]. Journal of CAD & CG, 1998, **10**(5):457 ~ 463. [袁斌,金其杰. 消除 Marching Cubes 二义性的新方法[J]. 计算机辅助设计与图形学学报. 1998, **10**(5):457 ~ 463.]

6　He Hui-guang, Tian Jie. A 3D medical imaging surface reconstruction scheme based on segmentation [J]. Journal of Software, 2002, **13**(2):219 ~ 226. [何晖光,田捷. 基于分割的 3 维医学图像表面重建算法[J]. 软件学报, 2002, **13**(2):

219 ~ 226.]

7 Wang Wei-xin Deng Da-hua. Research on visualization of arbitrary hexahedron elements data [J]. Journal of CAD & CG, 2000, **12**(8): 605 ~ 608. [王威信, 邓达华. 基于任意六面体单元数据场的可视化研究 [J]. 计算机辅助设计与图形学学报, 2000, **12**(8): 605 ~ 608.]

8 Liu Ying, Cai Kang-Ying. Progressive out-of-core compression based on reconstruction with marching cubes [J]. Chinese Journal of Computers, 2004, **27**(11): 1456 ~ 1463. [刘迎, 蔡康颖. 基于 Marching Cubes 重组的外存模型渐进压缩 [J]. 计算机学报, 2004, **27**(11): 1457 ~ 1463.]

9 Xie Xiao-mian, Li Shu-xiang. Smoothing and merging of medical isosurface based on MC algorithm [J]. Journal of Image and Graphics, 2001, **6**(8): 805 ~ 809. [谢小棉, 李树祥. 基于 MC 的医学 3 维等值面的平滑与归并 [J]. 中国图象图形学报, 2001, **6**(8): 805 ~ 809.]

10 Lin Chin-feng, Yang Don-lin. A marching voxels method for surface rendering of volume data [A]. In: Proceedings of IEEE International Conference on Computer Graphics [C], Hong Kong, 2001: 306 ~ 313.

11 Chen Mao, Tang Ze-sheng. Fast-cutting algorithm for 3D Surface model [J]. Journal of Software, 1998, **9**(9): 661 ~ 664. [陈矛, 唐泽圣. 三维表面模型的快速切割算法. 软件学报, 1998, **9**(9): 661 ~ 664.]

12 Tao Ju, Scott Schaefer. Convex contouring of volumetric data [J]. The Visual Computer, 2003, **19**(9): 513 ~ 525.