增量几何压缩+

刘新国 鲍虎军 彭群生
浙江大学 CAD&CG 国家重点实验室，310027，杭州

摘要 本文提出了一个几何压缩算法节省三角网格模型存储和传输时间。它首先递归地以区域扩张方式将模型分解为一系列的层结构，利用层间的连贯性，以及对层结构的有效编码，实现了高效的拓扑压缩。同时，还设计了一个有效的非线性预测器来实现几何位置的压缩。与以前的算法相比，它具有线性复杂度，压缩比高，执行速度快。实验结果表明，存储一个三角形的拓扑信息平均只需 1.42 比特。

关键词 几何压缩 二维流形 定向曲面 三角形网格模型

一、引言

尽管自由曲面广泛应用于CAD和计算机动画系统中，但多边形模型，尤其三角形网格由于其简单性和灵活性，也被大量的图形硬、软件普遍支持。近年来，稠密的三角形网格模型逐渐在许多应用领域，如数字地形、可视化、虚拟现实及基于三维扫描的自动造型中得到了越来越广泛地应用。这使得对高度复杂、精细三角形网格的实时编辑、绘制和传送成为一个极具挑战性的课题。最初人们为了减少绘制时间，加快绘制速度研究几何压缩算法[1,2,3,4,5]。

几何压缩不同于传统的图象压缩机制。一个几何网格模型通常由其拓扑和几何信息二部分组成，其中拓扑信息是指网格顶点之间的相互连接关系，而几何信息则指网格顶点的位置信息及附着在各顶点的有关绘制信息，如颜色、法向和纹理坐标等。几何压缩的目标是减少复杂三角形网格在其拓扑和几何位置信息表达方面的冗余度。

二、相关工作

类似于VRML中的IndexFaceSet，一个简单的三角形网格模型可表示为一个顶点数组和一个三角形数组，分别存放顶点的几何位置和三角形顶点下标。对于N个顶点和M个面的三角形网格模型，若用三个 4 个字节的浮点数来存贮每个顶点的空间坐标，4 个字节的整数表示顶点的下标，其存储量为 12N+12M字节。通常三角形个数是顶点个数的两倍左右，所以平均每个三角形的存储量是 18 字节。注意到这种表示方法中仍存在许多冗余信息，一种基于带状三角形结构（triangle strip）的三角形网格表示方法得到了图形学界的高度重视。通过重用前两个被访问过顶点和加入一个新顶点的方式来定义新的三角形，这种带状三角形结构减少了对顶点的索引次数，从而大大减少数据存储。在这种表示法中，平均而言，每一顶点被索引二次左右，存储一个三角形大约需 14 字节。尽管带状三角形结构可以减少对三角形网格模型的存储，但是采用一系列带状三角形结构完全覆盖一个具复杂拓扑的模型并非易事。进一步研究发现，采用更复杂的编码方法[6,7,8,9,10]，还能够大大地节省存储量。

Deering[2,5] 给出了一个一般化的带状三角形表示方法，通过增加索引缓冲的长度，在重用已访问顶点方面得到了更多的控制，从而减少数据存储量。Taubin等[8]在Topology Surgey算法中，通过构造一棵顶点树和一棵三角形树对连接关系进行编码。借助于这两棵树，表示一个三角形只需要 1 个比特。加上记录顶点树所需的额外数据，存储一个三角形平均仅需 1.2 至 3.5 比特(平均不超过 2 比特)。而Gumhold等[6]则构造和维护一些顶点序列组成的分割边界，并引入七种不同的操作构造下一个三角形，根据构造三角形所使用的操作对这些分割边界进行动态维护。由于这些操作的使用频率各不相同，且相差很大，所以可采用Huffman编码进一步减少数据量，存储一个三角形平均需要 1.5 至 4.23 比特。

多分辨表示和LoD简化算法[11,12,13,14,15,16,17]也可以看作是一类几何压缩算法，只是它改变了原来模型中顶点集合和拓扑信息，而这在很多情况下是不被允许的。但是Hoppe的累进网格方法[11,13]（简称PM）不同，它所表示的LoD模型序列具有很强的连惯性，通过简单的顶点分裂操作可以逐步地完全恢复原来的拓扑信息。再对顶点分裂操作的有效编码，就能实现

+ 本文收国家自然科学杰出青年基金（批准号：69925204），霍英东青年教师基金资助。

几何压缩。一个分裂操作包含一个分裂顶点和两条分裂边，如果当前层次模型的顶点数为N，存储分裂顶点需要Log(N)个比特；如果顶点平均入度为 6，存储两条分裂边平均需要大约 5.3 比特。Taubin 等[14] 扩展了PM方法，通过对一组相关的顶点分裂 ，称之为森林分裂(forest split)，进行编码，减少记录分裂操作的数据，从而获得更高的压缩比。拓扑信息压缩的关键在于充分地利用三角形顶点序列的连贯性。这需要我们以一种特殊地顺序和方式遍历网格中的三角形。通过以区域扩张的方法将原网格模型分解为一系列的层结构，在层与层之间保持极大的顶点连惯性，我们提出了一种有效的算法对拓扑编码。

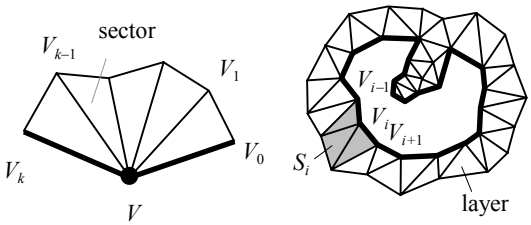
除了拓扑连接关系外，还需压缩几何信息。一般采用的策略是先量化、然后预测，最后存储校正码，量化的精度由用户根据要求选择。预测一般采用一个线行预测器。在Deering [2,5] 采使用了一个一阶线性预测器；而Taubin等在Topology Surgery算法[8]中利用使用了一个优化的高阶线性预测器，用最近访问的K个顶点来预测下一个顶点的空间位置。 因为校正值通常比较小而且相对非常集中，所以通常采用Huffman编码或算术编码。本文充分利用空间几何连贯性，设计了一种新的非线性预测器来预测下一个顶点的位置，其校正量则被定量在一局部坐标系中，并在用户给定精度范围内被量化并用Huffman编码。实验结果表明，本文所提出的非线性预测器是很有效的。

三、网格的表示

不失一般性 这里先假设模型是一定向、二维流形三角形网格，在第六节，我们将讨论一般非流形、非定向的三角形网格。一个三角形网格表示为一二元组<***V***，***T***>，其中***V***为其顶点集，***T***为其三角形集。***T***中的每个元素***t***表示为一有序三元顶点序列<V_0，V_1，V_2>，其顺序服从传统的右手规则。称顶点对<V_0，V_1>为一三角形的一条边，若存在一三角形***t***以V_0，V_1作为其顶点。我们称边<V_0，V_1>分别为顶点V_0和V_1的入边(incident edge) 。边<V_0，V_1>为网格的边界，若有且仅有一个三角形以V_0和V_1作为顶点。假设<V_0，V_1>为三角形网格的一条边，若<V_0，V_1，V_2>为网格上的一个三角形，则称该三角形为边<V_0，V_1>的左三角形; 若<V_1，V_0，V_2 >为一三角形，则称它为边<V_0，V_1>的右三角形。对一定向、二维流形三角形网格来说，其中的每条边有且至多有一个左三角形和一个右三角形。顶点序列 {V_0，V_1，…，V_{N-1}} 被称为网格的一条分割线，若<V_{i-1}，V_i>（i = 1，2，…，N，V_N=V_0）是网格上的边。显然，该分割线将原网格剖分成二部分。当沿该分割线前进时，称位于分割线左边的部分为其内部区域，另一部分为其外部区域。为了后面叙述方便，下面先定义一些基本结构。

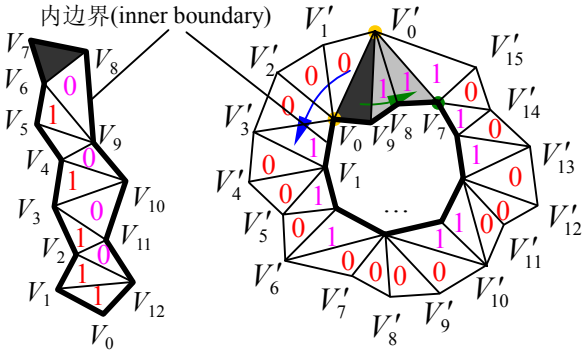
1．假设V_0，V和V_k是网格的三个相邻顶点，它们在网格上定义了一扇形区域Sector(V_0，V，V_k)，如图 1(a)所示，若网格上存在一顶点序列V_1 ，…，V_{k-1} 满足以下条件（如图 1(a)所示）：t_i = <V，V_{i-1}，V_i>（i = 1，2，…，k）均为网格上的三角形。{ t_i | i = 1，2，…，k} 构成了扇形区域的三角形集。

2．假设{V_0，V_1，…，V_{N-1}}为网格的一分割线，则它在网格上定义了一层结构Layer(V_0，V_1，…，V_{N-1})，如图 1(b)所示，若它满足：在该分割线的外部区域，V_{i-1}，V_i 和 V_{i+1}（i = 0，1，…，N–1）定义了扇形区域S_i，其中V_{-1} =V_{N-1}，V_N =V_0 。{S_i | i = 1，2，…，N–1}构成了该层的扇形集，其中的所有三角形构成了



(a) 扇形结构(sector)　(b) 层结构(layer)
(a) sector　(b) layer

图 1　基本结构
Fig 1　basic structures



(a) *L-strip*　(b) *O-strip*

图 2　*O-strip* 和 *L-strip*
Fig 2　*O-strip* and *L-strip*

该层的三角形集。分割线 $\{V_0, V_1, \ldots, V_{N-1}\}$ 称为该层的内边界线，若该层上还存在另一边界线，则称该边界线为该层的外边界线。

与原始网格相比，层结构复杂性大为降低，层分解是本算法的第一步。如图 2 所示，有两种非常简单的基本层结构，任何层都可以分解为这两种基本层结构的并。

3．第一种基本层结构称为 *L-Strip*，它可递归地定义为：

(1) *Layer*$\{V_0, V_1, V_2\}$为*L-strip*，若$<V_0, V_1, V_2>$为网格上的一个三角形。

(2) *Layer*$\{V_0, V_1, \ldots, V_{N-1}\}$若为*L-strip*，则当且仅当下列两个条件中有一个成立：

- $<V_0, V_{N-1}, V_1>$为一三角形且*Layer*$\{V_1, V_2, \ldots, V_{N-1}\}$为一*L-strip*；
- $<V_0, V_{N-1}, V_{N-2}>$为一三角形且*Layer*$\{V_0, V_1, \ldots, V_{N-2}\}$为一*L-strip*；

如图 2 (a)所示，实际上 *L-Strip* 类似于 OpenGL 中的带状三角形结构。特别地，只由一个三角形组成的 *L-Strip* 称为叶子（*leaf*）。

4．第二种基本层结构称为 *O-Strip*，如图 2(b)所示，一个层结构是 *O-Strip*，若对该层的三角形集中的每个三角形，至少有一个顶点位于其内边界线上，至少有另一顶点位于其外边界线上。

四、网格的分解与编码

本文压缩算法主要过程是以一种区域增量扩张的方式先将模型的顶点连接图分解为一系列的层结构，例如图 3；然后将层又分解为更简单的基本层 *O-strip* 和 *L-strip*；最后通过对两种 *strip* 压缩编码达到对整个拓扑关系的压缩。首先根据网格拓扑结构构造一个或多个层结构，如果有边界，我们就用边界线构造，否则选择一个三角形，用它 3 个顶点作为内边界构造初始的层，并将它们依次压入一个层堆栈。然后递归地每一步从层堆栈中弹出一个层结构作为当前处理层。若当前层既不是 *O-strip*，亦不是 *L-strip*，则将该层剖分为两个子层结构，分别压入层堆栈，并记录有关剖分信息到比特数据流中。否则，当前层为一基本带状三角形结构，我们写入一标识码到比特数据流中以指明该基本结构的类型，并紧随标识码后，将该结构的编码写入数据流中。若当前层为 *O-strip*，则还要抽取其外边界线，并由此构造一新的层压入层堆栈。上述过程直到网格被一系列层结构完全覆盖和编码为止。

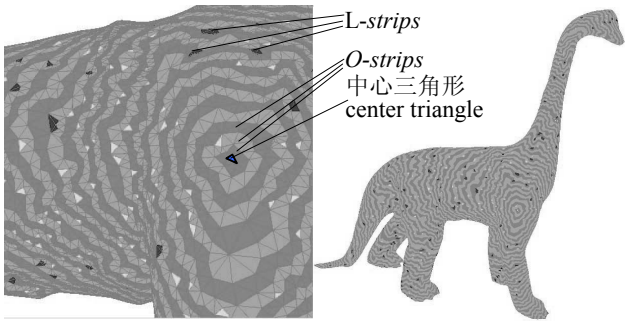


图 3 拓扑分解例子；明暗相间的环形区域为 *O-strip*；黑色区域为 *L-strip*；白色三角形为叶子。

Fig 3 Example of Topology Decomposition; the alternative gray and bright zones are *O-strips*; black regions are *L-strips;* white triangles are small leaves.

4.1 层的分解与编码

拓扑分解算法的关键之一是如何将当前层剖分为*O-strip*和*L-strip*，如图 4(a)、(b)所示。考虑刚从层堆栈中弹出的一层结构，记其内边界为 $\{V_0, V_1, \ldots, V_{N-1}\}$，算法从边$<V_0, V_{N-1}>$开始搜索其中的每个三角形，设$P$为边$<V_0, V_{N-1}>$的左三角形的第三个顶点，则下面二种情形必发生其一：

(1) 顶点P位于当前层的内边界上。这意味着在当前层中，一个子*L-strip*始于边$<V_0, V_{N-1}>$。（图 4(a)）算法进一步检测是否该*L-strip*恰好覆盖整个当前层。若是，则当前层为*L-strip*，按*L-strip*编码；否则，在当前层中存在一分枝，找到分叉点，并将当前层一分为二，记录剖分位置，并将两个子层结构压入层堆栈。

(2) 顶点P不位于当前层的内边界上。此时，说明在当前层中存在一*O-strip*始于边$<V_0, V_{N-1}>$（图 4(b)）。算法然后检测是否该*O-strip*恰好覆盖整个当前层。若是，则当前层为一*O-strip*，抽取其外边界线，由此构造一新的层，压入堆栈。否则，在当前层中存在一分枝，找到分叉点，将当前层一分为二压入堆栈，并记录剖分位置。

我们看到一个层结构可分为以下三类： *O-strip*，*L-strip* 和具有分支的层。一般地，需要 2 比特来标识这三种类型。注意到 *O-strip* 在拓扑分解产生的结构中占绝大多数，若用最

小的比特数来标识 *O-strip*，无疑会提高算法的效率。因此，我们采用下述方法来标识一层结构：若该层结构为 *O-strip*，用码 0 标识其类型，并写入比特数据流中，有关 O-strip 的编码将紧随该类型码后；否则，码 1 写入数据流，表明该层结构是一 *L-strip* 或具有分支。若它具有分支，码 1 后紧随两个变码长整数，表明两个分叉顶点在该层的内边界上的位置，从而提供相关的剖分信息。若当前层是一 *L-strip*，码 1 后紧随一大于内边界线上顶点数的不正常整数，然后是 *L-strip* 的编码。由于 *L-strip* 的数目极少，因而层类型的标识平均只需大约 1 比特。注意到当层结构具有分支时，产生的许多子层结构为叶子，既然一个整数足以确定每一叶子在层的内边界的位置，此时，算法可仅用一个整数而进一步优化编码效率。

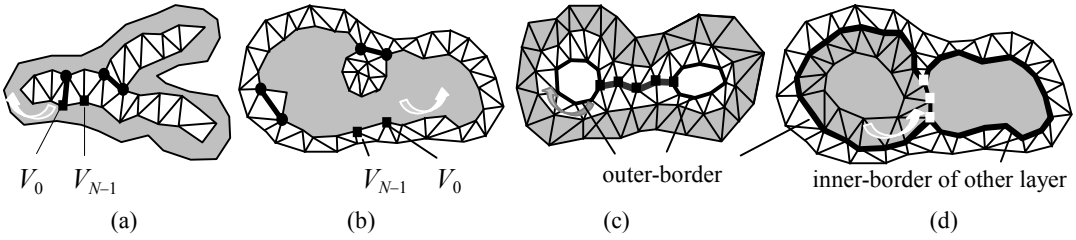


图 4 层分解及其编码。灰色区域表示已编码区域； (a) 该层被剖分为几个 *L-strip*，粗线和圆点表示分叉位置；(b) 该层被剖分为三个子层，粗线和圆点表示分叉位置；(c) 去掉粗线表示的桥，当前层的外边界分裂为两个边界，方形点表示近处引用顶点；(d) 当前层的外边界与层堆栈中的其它层的内边界合并，形成新的边界，白色点表示远处引用顶点。

Fig 4 Layer decomposition and encoding. The gray area represents the encoded area. (a) The layer is cut into several L-strips. The thick lines and the circle spots are the branching positions. (b) The layer is cut into three branches. The thick lines and the circle spots are the branching positions. (c) The outer-border of the layer is divided into two boundaries by cutting away the bridge (the thick line). The square spots are the near referenced vertices. (d) The outer-border (left thick line) is merged with an inner-border (right thick line) of other layer which still lies in the layer stack, and generate a new layer with the combined border(the whole thick line) as its inner-border. The white square spots are the far referenced vertices.

Algorithm (1): Encoding *the O-strip*

V_E is the last visited vertex during initialization process.
P_0 := *V_0*; *P_1* := *Q_0*; *stop-flag* := FALSE; *vertex-list* := { *Q_0* };
while *stop-flag is* FALSE **do**
  t := *the left triangle of edge* <*P_0* , *P_1*> ; *P_2* := *the third vertex of* *t*;
  if *P_0* = *V_E* *and* *P_2* = *Q_0* **then**
    write bit 1 *into the bit stream*; *generate new layers using vertex-list* {*Q_0*, *Q_1*,…,*Q_{M-1}*};
    push new layers onto layer stack; *stop-flag* := TRUE;
  else if *P_2 is the next vertex of P_0 on the inner-border* **then**
    write bit 1 *into the bit stream*; *P_0* := *P_2*;
  else if *P_2 is never visited before* **then**
    write bit 0 *into the bit stream* ; *write geometric information of vertex P_2 into the bit stream*;
    add P_2 to the tail of vertex-list; *P_1* := *P_2*;
  else if *P_2* = *Q_k* **then**
    write bit 0 *into the bit stream*; *write near-referenced vertex information into the bit stream* ;
    generate a new layer using {*Q_k* ,…,*Q_{M-1}*} *as inner-border* ; *push new layer onto layer stack* ;
    vertices Q_{k+1} ,…,*Q_{M-1} are removed from vertex-list* {*Q_0* ,…,*Q_{M-1}*}; *P_1* := *P_2* ;
  else if *P_2 lies on other layer's the inner-border* **then**
    write bit 0 *into the bit stream*; *write far-referenced vertex information into the bit stream* ;
    merge the far-referenced layer's inner-border into vertex-list ;
    remove the far-referenced layer from the layer stack ; *P_1* := *P_2* ;

4.2 *L-strip* 和 *O-strip* 编码

经拓扑分解后，所有产生的层结构均为基本带状三角形结构*L-strip* 的*O-strip*，下面我们将叙述它们的编码算法。对一条*L-strip*，参看图 2(a)，假设{V_0，V_1，…，V_{N-1}}为其内边界线，编码时，动态保持最近访问过的两个顶点P_0和 P_1，并找到边<P_0，P_1>的左三角形及其第三个顶点P_2。若P_2为P_0在内边界线上的下一顶点，则三角形<P_0，P_1，P_2>的编码为 1；否则，其编码为 0。注意到若在内部边界线上，P_0的下一顶点和P_1的前一顶点均为P_2，则结束*L-strip*的编码，最后一个三角形的编码可被省略。图 2(a)中*L-strip*的编码如下：

1 1 0 1 0 1 0 1 1 0

对于*O-strip*，因为它是一个首尾相接的封闭结构，为了避免重复遍历顶点，所以在编码之前要对它进行初始化。初始化过程从*O-strip*的内边界线上的边<V_0，V_{N-1}>开始，找到该边的左三角形及其第三个顶点V'_0，V'_0将成为该*O-strip*的外边界的第一个顶点。然后从V_{N-1}开始逆向遍历其内边界线，直至下一个顶点V_i不再与V'_0形成网格上的一条边为止。每个被访问的三角形的均被编码为 1，如图 2(b)中的灰色三角形。初始化结束则用码 0 来标识。

类似于*L-strip*，对*O-strip*编码也要记录最近访问的两个顶点P_0和P_1。除此之外因为*O-strip*有后继的层，所以对还要设置一个顶点链表，记录这个*O-strip*的外边界线 {Q_0，Q_1 ，…，Q_{M-1}}。编码过程中不断查找位于有向边<P_0，P_1>的左三角形t和这个三角形的第三个顶点P_2，然后根据顶点P_2是否位于外边界线上将三角形t编码为 0 或 1。当P_2 位于外边界的时候，紧跟着比特 0 后面写入外边界上顶点P_2 的信息。值得指出的是，此时顶点P_2可能是外边界顶点链表{Q_0，Q_1，…，Q_{M-1}}中一个曾经被访问过的顶点，即$P_2 = Q_k$，$0 < k < M$，或是层堆栈中另外一个层的内边界上的点，如图 4(c), (d)所示。这两种情况都需要进行特别处理。对于前一种情况，称P_2为近处引用点（near-referenced vertex），此时，该三角形被编码为 0，其后紧随该引用顶点在当前顶点链表中的索引号，然后以{Q_k，…，Q_{M-1}}为内边界线生成一个层，压入堆栈，同时当前的顶点链表更新为{Q_0 ，…，Q_k，Q_{M-1}}。在后一种情况中，称P_2为远处引用点（far-referenced vertex）。此时，将该引用点所在层的内边界融入当前的顶点链表中，并将被引用的层从堆栈中删去。例如图 2 (a)中O-strip外边界和三角形编码序列为：

V'_0 1 1 0 0 V'_1 0 V'_2 0 V'_3 1 0 V'_4 0 V'_5 1
0 V'_6 1 0 V'_7 0 V'_8 0 V'_9 0 V'_{10} 1 1 0 V'_{11}
0 V'_{12} 0 V'_{13} 1 0 V'_{14} 1 0 V'_{15} 1

其中前 4 个元素“V'_0 1 1 0”是初始化的结果。

4.3 顶点位置压缩

传统压缩方法首先将位置坐标在给定精度下量化，然后用一个线性预测器对顶点的位置进行预测校正，最后对校正项进行 Huffman 编码。以前的一些工作都是基于一串顶点位置的空间连贯性，对顶点位置进行线性预测。而我们的方法是利用局部几何的连贯性对顶点位置进行非线性预测，然后对预测的校正量在用户给定的精度内进行量化，并进行 Huffman 编码。

设<P_0，P_1，P_2>为当前处理三角形，P_2为其第三个顶点，在此之前P_2从未被访问过，则P_2可以分解为预测项和校正项的和

$$P_2 = \varepsilon(P_2) + P(P_0, P_1, \mathbf{N})$$

其中$\varepsilon(P_2)$是校正项，$P(P_0, P_1, \mathbf{N})$是对P_2的预测项，N为有向边<P_0，P_1>的右三角形所在平面H的法向。如果有向边<P_0，P_1>的右三角形并不存在，则取N为一个预先定义的固定向量。作为一个折衷的选择，我们取预测项$P(P_0, P_1, \mathbf{N})$为平面H上点，使得它和P_0，P_1两个顶点构成一个等边三角形。$\varepsilon(P_2)$是$P(P_0, P_1, \mathbf{N})$ 和 P_2之间的向量差（如图 5 所示）。注意到，根据几何的局部连贯性，$\varepsilon(P_2)$在法向N上的投影一般都很小。为了利用这一性质，我们在如图 5 所示的一个局部坐标系U-V-W中对$\varepsilon(P_2)$进行编码，其中U 是P_0到P_1的方向矢量，W等于法向量N，V 是垂直于U和W的第三个轴向量。$\varepsilon(P_2)$在W上的投影大小反映了曲面局部的平坦性，在其它两个坐标轴上投影的最大值由整个三角形网格上的最大边长决定。

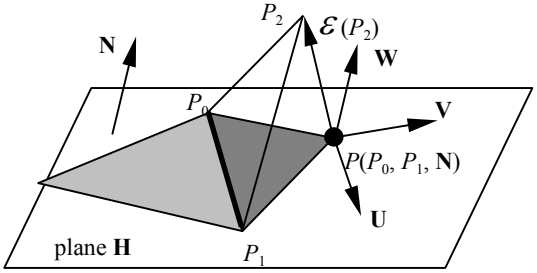


图 5 非线性预测器的几何说明
Fig 5 Non-LinearPredictor for gemoetry

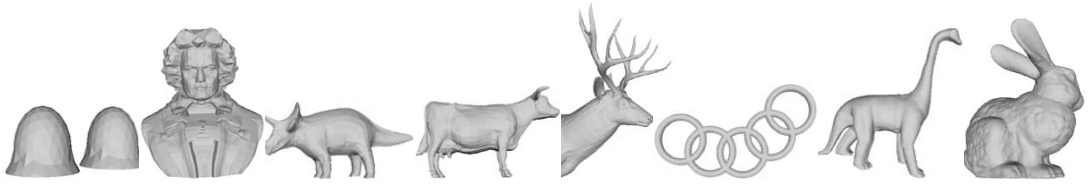
在 Huffman 编码的过程中，涉及到三种类型的顶点信息，第一种是由校正量表示的顶点坐标，另外两种是近处引用顶点和远处引用顶点的信息。为了在解码过程中能够识别这三种类型，必须用一种统一方式对它们进行编码。在目前实现的算法中，我们用 Huffman 表的第一个和最后一个入口分别表示近处引用和远处引用类型。在比特数据流中，对于顶点位置的信息，首先写入校正项的 W 分量编码，然后写入其它两个分量的编码。对于引用顶点

信息，首先写入引用类型的编码，然后是顶点的索引。顶点的索引则采用可变长整数表示以进一步提高编码效率。

解码是压缩的逆过程。解码之前首先从比特数据流中读出一些边界层，然后将它们压入层堆栈之中。然后从堆栈栈顶弹出一个层，对这个层解码，必要时产生一些新的层，并将它们压入堆栈。这个过程不断重复进行直到层堆栈为空。这样，整个网格被一层接一层，一个三角形接着一个三角形地被恢复出来。

五、实现及结果

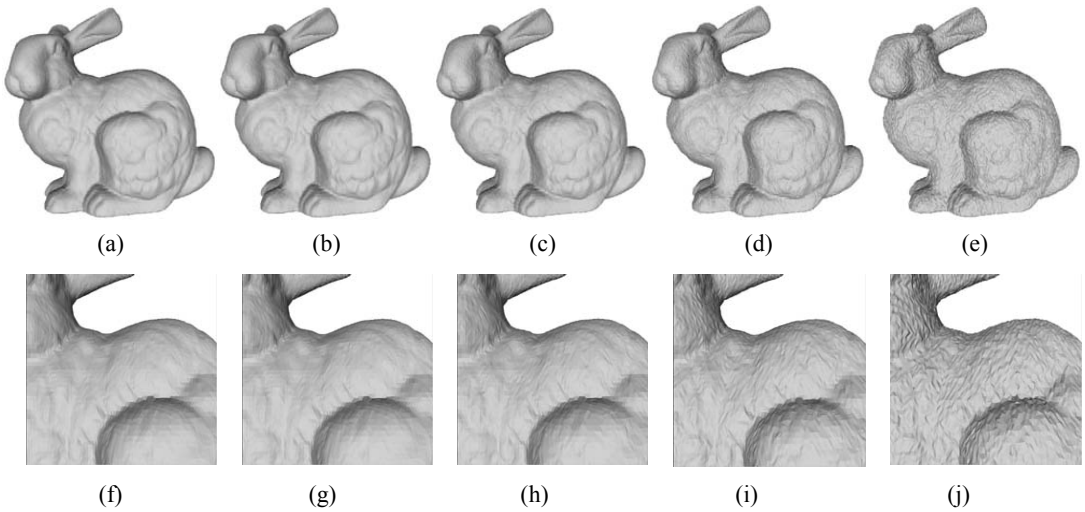
我们在 64M RAM 的 Compaq NT 工作站上用 C++实现了上述压缩和解压缩算法，并采用了八个不同类型的测试模型检验本算法的效率，有的是带柄的，有的是带边界的，具体见图 6。



Model 1 Model 2 Model 3 Model 4 Model 5 Model 6 Model 7 Model 8

	Model1	Model2	Model3	Model4	Model5	Model6	Model7	Model8
Vertex No.	310	2282	2555	3107	4413	9025	23931	34835
Triangle No.	540	4463	5094	6208	8822	18061	47858	69473

图 6 测试模型
Fig 6 testing models



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

图 7 不同精度的兔子模型。 (a)&(f): 原始模型；(b)&(g): 12 比特坐标精度；(c)&(h): 10 比特坐标精度；(d)&(i): 9 比特坐标精度；(e)&(j): 8 比特坐标精度。
Fig 7 The bunny model with different precision. (a)&(f): The original models. (b)&(g): 12 bits per coordinate. (c)&(h): 10 bits per coordinate. (d)&(i): 9 bits per coordinate. (e)&(j): 8 bits per coordinate.

对这些网格模型的压缩和解压缩的结果列于表 1。统计结果中包括顶点拓扑连接关系、位置压缩的效率、压缩速度以及解压的速度，其中速度根据机器的系统时钟统计而得。为了说明算法中非线性预测器的效果，我们尝试用一个十阶线性预测器对校正量再次预测以进一步减少顶点位置中的冗余信息。结果发现，校正项的相关程度很低，压缩效率没有明显的提高。

我们还对兔子模型用不同的精度进行压缩，观察不同的精度所引起的误差，压缩及解压的模型如图 7 所示。在表 1 中，我们发现：

(1) 拓扑连接关系的压缩数据总是低于 1.70 bit/triangle，平均是 1.42 bit/triangle。
(2) 在 10 bit 精度之下，顶点位置的平均压缩效率是 6.85 bit/triangle。模型的表面越光滑，压缩效率越好。
(3) 压缩的平均速度大约每秒 29757 个三角形，解码的平均速度大约每秒 24945 个三角形。总的说来，和前人的方法相比之下，对于连接关系和顶点位置，我们的压缩方法效率都要高且稳定。另外，本算法的执行速度亦很快。

表 1 压缩结果统计表(table of compression result)
Table 1 compression result statistics

		Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
8 bit precision	Connectivity (bits/tri)	1.28	1.55	1.64	1.38	1.37	1.44	1.58	1.14
	Position (bits/tri)	7.12	6.06	5.06	5.10	5.24	3.13	2.58	3.08
	Total (bits/tri)	8.40	7.61	6.70	6.48	6.61	4.57	4.16	4.22
10 bit precision	Connectivity (bits/tri)	1.28	1.55	1.64	1.38	1.37	1.44	1.59	1.14
	Position (bits/tri)	9.69	9.05	7.83	7.91	7.16	4.97	4.24	5.00
	Total (bits/tri)	10.97	10.60	9.47	9.29	8.53	6.41	5.83	6.14
12 bit precision	Connectivity	1.28	1.55	1.64	1.38	1.37	1.44	1.59	1.14
	Position	11.57	12.13	10.73	10.89	10.07	7.62	6.89	7.75
	Total (bits/tri)	12.85	13.68	12.37	12.27	11.44	9.06	8.48	8.89

六、扩展

有的几何模型还包含顶点和三角形的绘制信息，例如颜色和纹理坐标、法向等。我们知道，所有的这些信息都可以表示为 3D 或 2D 的矢量，因此，可以采用与压缩顶点位置信息类似的方法进行压缩，不同的是针对它们各自的特点设计相应的预测器

对于非定向或非流形三角形网格，我们的解决方法是在预处理阶段，将网格分解为几个定向的子流形网格，然后对每一个定向子流形网格用上述的方法分别进行压缩编码。

如果原始网格不是二维流形，通过检查每个顶点同它的相邻顶点的连接关系，找到那些奇异点，然后分裂并记录这些奇异点，将原始网格分解为一些子流形网格。类似地，非定向网格也可以分解为一些子定向网格。我们可通过网格上局部定向的传播，确定所有顶点和三角形的定向，检测出所有定向冲突的边，即其左右两个三角形具有相反方向。然后将这些边分裂，使得其左右两个三角形不再共享这条边，最后得到一些子定向二维流形网格。

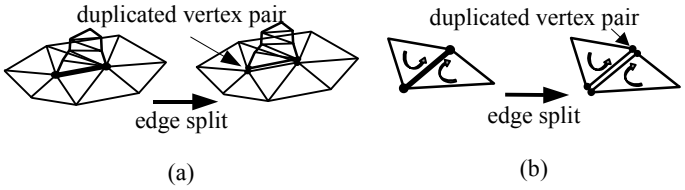


图 8 非流形和非定向网格上边的分裂，加粗边和圆点为奇异边和顶点。(a) 非流形情形； (b) 非定向情形。
Fig 8 The edge splits in non-manifold and non-orientable meshes. The thick edges and spot vertices are the extraordinary edges and vertices. (a) The non-manifold case. (b) The non-orientable case.

很显然，将原始的网格分解为定向的二维流形之后，它的拓扑已经发生变化了。如果我们想要压缩之后保持原始的拓扑性质，必须保存顶点复制的信息，并将它们添加到压缩文件的末尾，以便解压缩最后阶段利用这些信息将复制后的顶点合并起来去准确恢复拓扑性质。

七、结论

本文描述了一个非常高效的压缩、解压缩算法，该方法充分利用拓扑连接关系的连贯性，以区域增量扩张的方式将三角形网格分解为一系列的层结构，每一个层又进一步被分解为一些能够被高效编码的基本带状三角形结构，从而取得拓扑信息的高效压缩。另外，我们利用几何的局部连贯性，提出了一个非线性预测器来减少顶点位置之间冗余信息。正如文中所述，

和以前的工作相比，本算法能够获得非常高而稳定的压缩效率，且压缩和解压缩过程具线性复杂度，能以很快的速度执行。

参考文献

[1] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. S. Skiena. Hamiltonian Triangulations for Fast Rendering. Lecture Notes in Computer Science, 1994, 23(1) : 36~57.

[2] M. Deering. Geometrical compression. SIGGRAPH'95 Proceedings, Edited by Robert Cook, Los Angeles, 1995, 13~20.

[3] R. Bar-Yehuda and C. Gotsman. Time/Space Tradeoffs for Polygon Mesh Rendering. ACM Transactions on Graphics, 1996, 15(2) : 141~152.

[4] F.Evans, S. Skiena, A.Varshney. Optimizing triangle strips for fast rendering. In IEEE Visualization '96 Proceedings, Editted by Roni Yagel etc., San Francisco, 1996, 319~326.

[5] M. Deering. Hardware Geometrical compression Specification from Java 3D. SIGGRAPH'98 Course, #21, Edited by Michael Cohen etc., Orlando Florida, 1998, 296~320.

[6] S. Gumhold and W. Straβer. Real Time Compression of Triangle Connectivity. SIGGRAPH'98 Proceedings, Edited by Michael Cohen etc., Orlando Florida, 1998, 133~140.

[7] G. Taubin, W. P. Horn, F. Lazarus, and J. Rossignac. Geometric Coding and VRML. IBM Research Technical Report RC-20925, July 1997

[8] G. Taubin, J. Rossignac. Geometrical compression through Topological Surgery. IBM Research Technical Report RC-20340, January 1996.

[9] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. GVU Technical Report GIT-GVU-98-35, 1998

[10] V. Bajaj, V. Pascucci and G. Zhuang. Single Resolution Compression of Arbitrary Triangular Meshes with Properties. Edited by Michael Cohen, Proceedings of Data Compression Conference, Snowbird, Utah, 1999.

[11] H. Hoppe. Progressive Meshes. SIGGRAPH'96 Proceedings, Edited by Holly Rushmeier etc., New Orleans, Louisiana, 1996, 99~108.

[12] D. Zorin, P. Schroer, and W. Sweldens. Interactive Multiresolution Mesh Editing. SIGGRAPH'97 Proceedings, Edited by John M. Snyder, Los Angeles, 1997, 259~26.

[13] H. Hoppe. Efficient Implementation of Progressive Meshes. Computers & Graphics, 1998 22(1) : 27~36.

[14] G. Taubin, A.Gueziec, W. Horn, and F. Lazarus. Progressive Forest Compression. SIGGRAPH'98 Proceedings, Edited by Michael Cohen etc., Orlando Florida, 1998, 123~132.

[15] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution Analysis of Arbitrary Meshes. SIGGRAPH'95 Proceedings, Edited by Robert Cook, Los Angeles , 1995, 173~182.

[16] A. Certain, C. Popovic, T. DeRose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive Multiresolution Surface Viewing. SIGGRAPH 96 Proceedings, Edited by Holly Rushmeier etc., New Orleans, Louisiana, 1996, 91~98.

[17] M. Lounsbery, T. DeRose, and J. Warren. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. Transactions on Graphics, 1997, 16(1) : 34~43.

Incremental Geometrical Compression

Liu Xinguo Bao Hujun Peng Qunsheng

Abstract In this paper, a geometry compression algorithm is presented to save the geometry model storage and transmission time. This method decomposes the model into a serial of layers in a region growing way. These layers are then encoded effectively by taking advantage of inter-layers coherence, so that the topology information of the model is compressed dramatically. Additionally, a non-linear geometry predictor is designed to compress the geometric information. Compared with the previous works, this algorithm is of linear complexity, higher compression ratio and very fast. Experiments showed that it takes only an average 1.42bits per triangle.

Keywords: Geometrical Compression, 2-manifold, orientable surface, triangulated model.