ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.2, February 2008, pp.446−454
DOI: 10.3724/SP.J.1001.2008.00446

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于上下文的网格拓扑压缩熵编码方法*

刘 迎[1,2+], 刘学慧[1,2], 孙春娟[1,2], 吴恩华[1,2,3]

[1](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100080)

[2](中国科学院 研究生院,北京 100049)

[3](澳门大学 科学技术学院 电脑与资讯科学系,澳门)

Context-Based Entropy Encoding Method for Connectivity Compression of Meshes

LIU Ying[1,2+], LIU Xue-Hui[1,2], SUN Chun-Juan[1,2], WU En-Hua[1,2,3]

[1](Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[3](Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macao, China)

+ Corresponding author: E-mail: liuyingbj@yahoo.com

Liu Y, Liu XH, Sun CJ, Wu EH. Context-Based entropy encoding method for connectivity compression of meshes. *Journal of Software*, 2008,19(2):446−454. http://www.jos.org.cn/1000-9825/19/446.htm

Abstract: A general efficient algorithm for entropy encoding of the connectivity information of meshes is presented in this paper. In comparison to the previous encoding methods, which use only Huffman or arithmetic coding method to encode operator series, this coding method can efficiently compress connectivity information by first calculating Huffman code for every symbol in connectivity series, followed by encoding the Huffman code through using a context-based arithmetic coding method. Experimental results indicate that this method can be applied to almost all the connectivity compression algorithms for meshes. The compression result by using this entropy encoding method is generally higher than the entropy of the series—the best compression result that most connectivity compression algorithms of mesh can obtain respectively.

Key words: Huffman code; context-based arithmetic coding; mesh; connectivity compression; encode; decode

摘 要: 提出了一种普遍适用于网格拓扑压缩的高效熵编码方法.不同于以往的单纯利用算术编码或 Huffman 编码对遍历网格生成的拓扑流进行编码压缩,对这些拓扑流的每个符号先计算其 Huffman 编码,然后采用基于上下文(已编码序列的倒数第 2 个符号作为上下文)的算术编码方法来编码其 Huffman 值,从而实现对网格模型拓扑信息的有效压缩.实验结果表明,熵编码方法普遍适用于各种网格拓扑压缩方法得到的拓扑流的压缩,其压缩结果普遍高于拓扑流序列的熵值——绝大多数拓扑压缩算法各自最好的压缩比.

关键词: Huffman 编码;基于上下文的算术编码;网格;拓扑压缩;编码;解码

中图法分类号: TP393 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60373051, 60173022 (国家自然科学基金); the National Basic Research Program of China under Grant No.2002CB312102 (国家重点基础研究发展计划(973)); the China Research Grant of University of Macau (澳门大学研究基金)

Received 2006-08-16; Accepted 2006-11-30

随着计算机图形学技术在现实生活中的普及,计算机图形学的研究领域越来越广泛.近些年,越来越多的三维网格数据在各应用领域,如电子商务、医疗、科学计算可视化、工程分析和游戏等领域随处可见.三维网格数据具有数据量庞大的特点,这使得网格压缩算法的研究在计算机图形学领域占有很重要的地位.

如果从类型来分类三维网格数据,则有几何数据(如顶点坐标)、拓扑数据(顶点间是否有边连接等信息)和属性数据(如颜色、法向量、纹理等信息).对这些类型的数据作高效的压缩是网格压缩算法研究的内容.在目前的网格压缩算法中,用于压缩几何数据的算法称为几何压缩算法.因为属性数据和几何数据性质相似,所以目前都采用几何压缩方法来压缩属性数据.对拓扑数据作压缩的算法称为拓扑压缩算法.所以,目前的网格压缩算法如果从处理数据类型来分类,只分为几何压缩算法和拓扑压缩算法.并且,有的网格压缩算法能够同时压缩所有类型数据,在分类上一般就把该算法同时归为两类,如文献[1]中的算法.

对于所有的几何压缩算法,其压缩过程都分成两个部分:1) 量化数据;2) 预测编码数据并编码预测值和实际值的差.本文研究网格拓扑压缩算法,关于几何压缩算法,参见文献[1–3].而拓扑压缩算法也是有共性的:1) 先用某种网格遍历方法得到操作符序列;2) 采用某种熵编码来压缩操作符序列得到算法的最终压缩结果.关于网格压缩的更多内容,可参见文献[4–6].

本文的算法属于网格拓扑压缩算法.本文在各种网格遍历算法的基础上,提出一种具有普遍性的熵编码方法,也就是说,本文尝试提出一种适合所有网格拓扑压缩算法的熵编码方法.现有的拓扑压缩算法大多采用 Huffman 编码[7](早几年)或算术编码[8](近几年),算术编码的结果要稍好于 Huffman 编码结果.本文的熵编码算法不是 Huffman 编码,也不是算术编码,而是二者的组合:先用 Huffman 编码,再利用基于上下文的算术编码来编码 Huffman 结果.用本文的方法分别压缩各种网格拓扑压缩算法中的拓扑流,其压缩比要好于直接采用算术编码得到的压缩比,更要好于用 Huffman 方法得到的压缩比.

本文第 1 节介绍与本文算法相关的工作.第 2 节给出本文的熵编码压缩算法.第 3 节给出实验结果.第 4 节为结论.

1 相关工作

1.1 网格拓扑压缩算法概述

本文尝试提出一种对所有网格拓扑压缩算法普遍适用的熵编码方法.拓扑压缩算法中的熵编码是对各种拓扑流作编码压缩,不同的网格拓扑压缩算法中的拓扑流(通过网格遍历方法得到)是不同的.根据网格遍历算法过程中主要操作单元的不同,网格拓扑压缩算法分为基于顶点[1,9,10]、基于边[11]、基于面[12−17]等不同类型.现有的大部分网格拓扑压缩算法属于基于顶点和基于面这两类,并且另一类基于边的网格拓扑压缩算法和基于面的拓扑压缩算法很相似,所以,本文研究基于顶点和基于面这两类算法,在一定意义上可以代表网格拓扑压缩领域的所有算法.

最早的基于顶点的拓扑压缩算法是 Touma 和 Gotsman 于 1998 年提出的针对三角网格拓扑信息作单分辨率无损压缩的算法[1].该算法仍然是到目前为止压缩效果非常好的算法之一.该算法沿螺旋顶点树遍历并编码每个顶点,输出每个顶点的邻接三角形个数作为操作符,分支处需记录额外的信息.该算法用算术编码[8]压缩操作符序列.对于规则的三角网格,该算法压缩效果极好.Alliez 等人[9]在该算法基础上,通过自适应遍历网格,使很影响压缩比的分割(split)图形的操作尽可能地少,从而稍稍提高了压缩比.Khodakovsky 等人[10]在该算法基础上扩展其应用,使其能够压缩多边形网格,不再局限为只能处理三角形网格.该方法用文献[1]所提出的算法中的网格遍历方法遍历原网格模型及其对偶(dual)[4]网格模型,可分别获得两个拓扑流.然后用基于上下文的算术编码分别压缩这两个拓扑流,其压缩比与直接应用算术编码相比平均提高约 13%.

最早的基于面的拓扑压缩算法是 Rossignac 于 1999 年提出的 Edgebreaker 算法[16].该算法针对三角网格拓扑信息作单分辨率无损压缩,是一种经典的算法,并且与文献[1]中的算法在遍历网格方法上很相似,两者只是遍历网格后输出的操作符不同,文献[1]中的算法输出每个顶点的邻接面个数,而文献[16]中的算法输出每个面与已遍历网格部分的拓扑关系.在编码遍历网格的输出结果方面,文献[16]中的算法用 Huffman 编码[7],而文献[1]

中的算法用算术编码.该算法提出来以后,有多种算法[13−15,17]对其作了改进.其中较新的是 Jong 等人提出的对 Edgebreaker 算法的网格遍历过程加以改进的算法[14],改进算法用 J 替代很影响压缩比的 S 操作,使编码字符由 CLERS 改变为 QCRLJ,并且使得解码简单而且是一次遍历.该算法采用算术编码(Edgebreaker 采用 Huffman 编码方法)来直接压缩 QCRLJ 编码字符序列,极大地提高了压缩比.Kronrod 等人[18]提出的算法和 Lee 等人[19]提出的算法是对 Edgebreaker 算法的直接扩展,可以压缩多边形网格模型.两者略有不同的是,文献[18]中的算法使用 Huffman 方法压缩操作符序列,而文献[19]中的算法使用算术编码方法.

1.2 网格拓扑压缩算法过程

网格拓扑压缩算法有基于顶点、基于边和基于面 3 类,但实际上,这 3 类方法有很多共性:先遍历网格模型获得操作符序列;再采用某种熵编码方法来压缩这个操作符序列.

在网格拓扑压缩中,基于面和基于边的网格遍历方法有相似的原理.其过程简单叙述如下:

1) 先任选一个面片形成初始边界,并且指定边界上的一条边为当前边.该面片标记为已遍历.

2) 寻找包含当前边但未被遍历的面片,并对该面片作遍历操作(标记该面片为已遍历)或跳过该面片(继续标记该面片为未遍历),同时更新边界和当前边.不同的算法对该操作记录不同的操作符,它们更新边界和当前边的规则也不同.

3) 重复 2),直到所有面片都标记为已遍历,算法遍历网格过程结束,此时得到操作符系列.

在网格拓扑压缩中,基于顶点的网格遍历方法与上述方法很相似,但也有不同.其过程简述如下:

1) 先任选一个面片形成初始边界,并且指定边界上的一条边为当前边.输出面片上每个顶点的邻接面个数.标记这些顶点和这个面片为已遍历.

2) 寻找包含当前边但未被遍历的面片,输出该面片上新增顶点的邻接面个数,并标记这些顶点和这个面片为已遍历,同时更新边界和当前边.

3) 重复 2),直到所有面片和顶点都标记为已遍历,算法遍历网格过程结束,得到所有顶点的邻接面个数序列.

无论哪种网格拓扑压缩算法,在得到遍历网格的输出序列(操作符序列)以后,都要采用某种熵编码——Huffman 或者算术编码来压缩这个序列,得到网格拓扑压缩的最后压缩结果.

1.3 熵编码

对于给定具体序列,采用某种优化编码方法对其编码后,每个操作符平均占用的比特数定义为熵.熵可以为小数,并且这个熵值有一个理论值[5]: $-\sum_{i=1}^{n} p_i \cdot \log_2 p_i$,其中,n 为操作符种类,p_i 为第 i 种操作符的概率,且 $\sum_{i=1}^{n} p_i = 1$. 如果只有具体序列而没有任何其他信息,则上述熵理论值是能够达到的最好的压缩结果[5].

整数方法实现的算术编码[8]的压缩结果可以达到熵理论值[5].从纯序列的编码压缩上看,Huffman 编码方法以整数逼近序列的熵值,而算术编码以小数逼近序列的熵值.从实践上看,算术编码的压缩结果一般情况下能够达到序列的熵值,误差一般不会超过 2%.如果对算术编码和 Huffman 编码作比较,则算术编码结果要好于 Huffman 编码结果.

与网格拓扑压缩有关的熵编码目前只有 Huffman 编码和算术编码这两种方法,而本文针对网格拓扑压缩的熵编码部分提出了一种高效的、普遍适用于网格拓扑压缩的新的熵编码算法.新算法融合了上述两种编码方法,先对序列作 Huffman 编码,然后对 Huffman 编码结果作基于上下文的算术编码,压缩结果好于序列的熵值.

2 本文的熵编码压缩算法

本文提出的算法是针对网格拓扑压缩领域的.该算法分两步完成:1) 利用某种网格拓扑压缩算法中的网格遍历方法遍历网格模型,得到操作符序列;2) 对操作符序列的每个操作符作基于上下文的算术编码:先计算该操作符的 Huffman 编码;然后用基于上下文的算术编码方法来压缩这个 Huffman 编码.

2.1 获取操作符序列

本文的算法要压缩网格拓扑流——操作符序列,所以首先要获得操作符序列.我们没有提出新的网格遍历方法,而是直接应用现有的各种网格拓扑压缩算法中的网格遍历方法.因为要提出的是一种普遍适用于网格拓扑压缩算法的熵编码,所以我们共实现了 6 种在网格拓扑压缩领域具有代表性的网格遍历方法来获取操作符序列,分别为:Touma 等人[1]的算法(简称 TG 算法)、Alliez 等人[9]的算法(简称 PA 算法)、Khodakovsky 等人[10]的算法(简称 KA 算法)、Rossignac[16]的 Edgebreaker 算法(简称 EB 算法)、Jong 等人[14]的算法(简称 JBS 算法)、Kronrod 等人[18]的算法(简称 KB 算法).

对于不同的网格遍历算法,其操作符序列的基本操作符数量是不同的,甚至同一种网格遍历算法针对不同的网格模型,其基本操作符数量也是不同的.下面是上述 6 种算法的基本操作符表示和概率分析.

TG 算法:基本操作符主要是每个顶点的邻接面个数,所以有一定的变化范围.另外,在 TG 算法中,还有少量的“分隔 S”及少量的“合并 M”操作,可分别用数字 0 和 1 表示,在这两个操作的后面,还要分别记录一个偏移量(也是一个数字).对于规则的三角网格模型,绝大多数顶点的邻接面个数为 6;对于一般常用的三角网格模型,大多数顶点的邻接面个数为 6.也就是说,在 TG 算法的操作符序列中,一般情况下,6 的概率最大,其次为 5 或 7,其他的数字的概率就很小了,但数字的范围比较大(基本操作符个数多).

PA 算法:基本操作符与 TG 算法的基本操作符相同.唯一不同的是:PA 算法优化确定当前边,这样就使得 PA 算法的操作符序列中的“分割 S”操作数量非常少(相比 TG 算法),使得输出的操作符个数减少,也因此,PA 算法能够得到比 TG 算法更高的压缩比.

KA 算法:基本操作符与 TG 算法的基本操作符一样.不同的是:KA 算法输出两个操作符序列,一个是原网格模型的,另一个是原网格的对偶网格模型[4]的.因为 KA 算法压缩非三角网格模型,所以,其输出的操作符序列的基本操作符不再像 TG 算法那样以 6 为多,而是根据不同的网格模型有很大的概率变化范围.

EB 算法:基本操作符为“增加 C”、“左边 L”、“右边 R”、“分割 S”、“封闭 E”.其中,C 的概率最大,其次为 R,L 的概率比较小,E 的概率更小.对于不同的三角网格模型,S 的概率是不同的,可能为 0,也可能多些,但总的来说,S 的概率不会很大.

JBS 算法:基本操作符为“增加 C”、“左边 L”、“右边 R”、“跳跃 J”、“四边形 Q”.对于三角网格模型的操作符序列,Q 和 C 的概率大,二者的概率相差不大,且二者的概率之和接近 0.95.

KB 算法:一般情况下,基本操作符个数大于 18.其中的 18 个基本操作符用于记录三角形和四边形与边界的拓扑关系.另外,还会有一个基本操作符用于表示边数大于 4 的多边形,该操作符后要记录边数大于 4 的多边形的边数及其与边界的拓扑关系索引.对于 KB 算法的操作符序列,Q2,Q5,Q15 的概率最大,三者概率之和接近 0.9.

2.2 熵编码

采用某种网格遍历方法得到操作符序列后,我们就可以对这个序列作熵编码.先是 Huffman 编码,然后是基于上下文方法得到中间序列——比较序列,最后对中间比较序列作自适应算术编码.

2.2.1 Huffman编码

根据具体的一个给定的操作符序列,其中每个基本字符有各自的概率分布属性.根据这个概率分布,我们可以为每个基本字符计算 Huffman 编码——固定的一个 0,1 序列,并且其长度是不同的.其 Huffman 编码的计算采用成熟、经典的算法[7].

为了保证正确解码,在压缩具体的操作符序列之前,我们必须先压缩每个基本字符的 Huffman 编码值.对于一个具体的网格模型,其操作符序列的基本字符数是一定的,且无论这个网格模型的规模如何,这一部分所占用的字节数都变化不大,而且占用字节数比较少.所以,对于大的网格模型(也是更需要高效压缩的模型),这部分字节对最后的压缩比影响非常小.

2.2.2 基于上下文方法

对于一个给定的操作符序列,知道了每个基本操作符的 Huffman 编码以后,我们要用基于上下文的方法给

出中间比较序列——最后的算术编码就要压缩这个比较序列.

对于一个给定的序列,它具有一个熵值,Huffman 编码方法以整数逼近序列的熵值,而算术编码以小数逼近序列的熵值.不论是直接采用算术编码方法压缩序列,还是间接地采用算术编码压缩序列的 Huffman 编码序列,其压缩结果最好时也只能是接近序列的熵值,而不会好于这个熵.所以,如果要达到更高的压缩比(高于序列的熵值),则必须改变序列.本文的中间比较序列就是不同于给定操作符序列的新序列,并且中间比较序列的熵要小于其相应的操作符序列的熵,也正因如此,才有了本文熵编码方法的高效压缩结果.

对于序列的每个操作符,本文实际上是按照序列顺序对每个操作符一个个地作处理以得到中间比较序列.当处理序列中的一个操作符时(简称当前操作符),我们以其前面已编码操作符部分的倒数第 2 个作为上下文,为了便于叙述,称其为预测操作符.当前操作符和预测操作符都有各自的 Huffman 编码值,如果当前操作符和预测操作符相同,则其 Huffman 编码具有相同的码值和码长;如果二者不同,则其 Huffman 编码具有不同的码值和可能不同的码长.

本文的中间比较序列的计算过程如下:对于当前操作符,我们逐个处理其 Huffman 编码的每个比特(bit).对于每个比特,我们将它与预测操作符的相应位置处的比特作比较,如果两个比特相同,则输出比特 0 到中间比较序列;如果不同,则输出 1 到中间比较序列.如果预测操作符的相应位置处没有比特值(预测操作符的 Huffman 码长小于当前操作符的 Huffman 码长),则与比特 0 作比较.

通过上述方法得到中间比较序列,其长度等于操作符序列的 Huffman 编码序列的编码长度.但与 Huffman 编码序列不同的是,把倒数第 2 个已编码操作符作为上下文的一个比较系列.图 1 给出了一个操作符采用基于上下文方法得到中间序列的示例.

实际上,我们实验了很多个其他已编码操作符作为上下文(预测操作符),当预测操作符 Huffman 码长小于当前操作符码长时,与比特 1(图 1 中增加的比特)作比较得到中间比较序列.实验结果表明,对于个别少量实验模型,某种上下文和增加比特的组合可能会稍好于本文采用的方法,但对于我们的所有实验模型,采用本文的方法能够普遍得到较好的压缩结果.

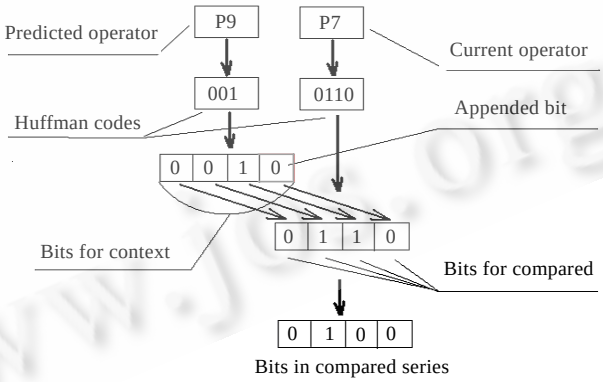

Fig.1 An illustration of calculating median compared series of an operator

图 1 一个操作符的中间比较序列计算示例

2.2.3 自适应算术编码

我们得到中间比较序列后要对其作自适应算术编码.从中间比较序列的计算过程可以看出:如果当前操作符与预测操作符相同,则其 Huffman 编码具有相同的码值和码长,其相应的中间比较序列都是 0,且 0 的个数等于其 Huffman 编码码长;如果二者不同,其 Huffman 编码具有不同的码值和可能不同的码长(如图 1 所示).从图 1 中可看出,当前操作符和预测操作符虽然不同,但当前操作符的相应中间比较序列中 4 个比特有 3 个比特是 0.这样的序列很适合算术编码.当前操作符和预测操作符不同时会有部分比特相同,这一点是不容置疑的.

本文算法的算术编码过程涉及到的具体内容有:

1) 对中间比较序列的每个比特作自适应算术编码.

2) 自适应算术编码采用整数实现方法[8].

算术编码结束后,就得到了本文算法的最终压缩结果,见第 3 节实验结果部分.

本文提出的新的熵编码方法同时用到了 Huffman 编码和算术编码,但实际上,我们并没有通过对操作符序列作 Huffman 编码来得到 Huffman 编码序列,而只是利用了操作符序列中的基本操作符的 Huffman 编码,包括计算基本操作符的 Huffman 编码和 Huffman 编码在内存中的驻留和读取.因为对于操作符序列来说,基本操作符数量是极其有限的,最多几十个,最少可以为 5 个.所以,本文虽然采用了 Huffman 编码,但在算术编码过程中,因为 Huffman 编码的介入所引起的时间和空间的额外消耗非常小,基本可以忽略不计.

2.3 解 码

网格解码过程由网格编码(遍历)过程决定.本文的熵编码过程不影响网格遍历过程,所以,各种网格遍历方法所具备的各自的优点仍然存在.特别需要提及的是,本文的熵编码方法不影响网格解码过程,只是略有不同.

在压缩某一个操作符时,我们要将前面的已编码操作符部分的倒数第 2 个操作符作为上下文(预测操作符),并根据最先传递的 Huffman 编码基本信息(序列基本字符的 Huffman 编码)来计算中间比较序列并压缩这个比较序列.而这些信息对于解码器来说也是已知的,解码时可根据同一个上下文、Huffman 编码基本信息以及解码得到的中间比较序列来得到当时的编码操作符.

解码时得到的是一个个 0 或 1 比特,所以解码一个比特时,如果该比特不是一个 Huffman 编码码值,则接着再解码下一个比特,两者组合如果还不是一个 Huffman 编码码值,则再接着解码下一个比特,直到出现一个 Huffman 编码组合——代表一个具体的操作符.此时根据出现的操作符,可直接利用某种网格解码方法解码面片或顶点.以此类推,可以很容易地正确解码整个操作符序列,从而正确解码原始网格模型.本文的熵编码/解码方法在保持拓扑压缩算法的优点的同时,但却能提高拓扑压缩算法的压缩比.

3 实验结果

我们在普通 PC 机上实现了 6 种网格拓扑压缩算法中的网格遍历方法(第 2.1 节获取操作符部分).这 6 种网格遍历方法在网格拓扑压缩领域具有代表性.对这 6 种网格遍历得到的操作符序列,采用本文的熵编码方法作压缩可得出具有普遍性的结论:本文的算法普遍适用于网格拓扑压缩算法的熵编码部分,能够提高这些算法各自的压缩比.

本文对三角形网格模型和多边形网格模型分别作了实验.实验用的部分三角形网格模型如图 2 所示,部分多边形网格模型如图 3 所示.

Fig.2 Partial test triangular mesh models

图 2 部分测试用三角形网格模型

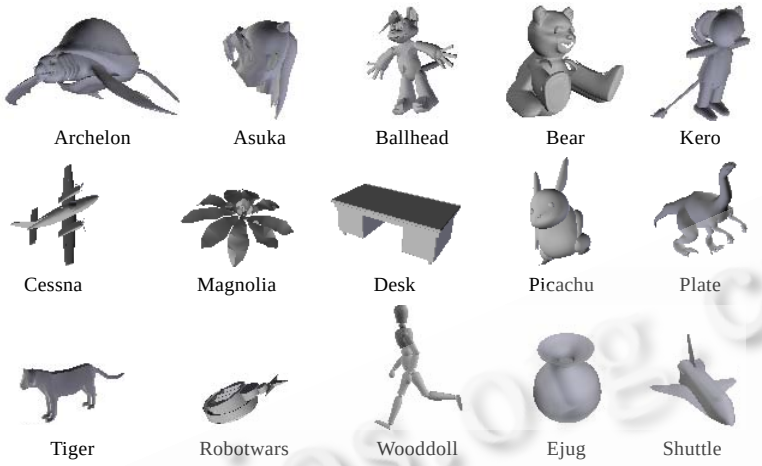

Fig.3 Partial test non-triangular mesh models

图 3 部分测试用多边形网格模型

三角形网格实验结果见表 1,多边形网格实验结果见表 2,其中,#V 表示网格顶点个数,#F 表示网格面片个数.表中 TG,PA,EB*,JBS,KA,KB 这 6 列为网格拓扑压缩各算法的操作符序列的自适应算术编码压缩比,这也是大部分原算法的实验数据(EB,KB 采用 Huffman 编码,其压缩结果稍差于自适应算术编码结果;KA 采用基于上下文的算术编码,其压缩比好于自适应算术编码压缩结果——平均提高 13%左右;其他 3 种算法都采用自适应算术编码方法);TG+,PA+,EB+,JBS+,KA+,KB+这 6 列为使用本文的熵编码方法压缩各算法的操作符序列的实验结果;%TG,%PA,%EB,%JBS,%KA,%KB 这 6 列为采用本文的熵编码方法压缩各算法的操作符序列得到的压缩比与采用自适应算术编码方法得到的压缩比相比所提高的百分比.

从表 1 和表 2 的实验数据以及实验算法在网格拓扑压缩领域的代表性可以看出,本文的熵编码方法能够普遍应用在网格拓扑压缩领域.其对各算法的操作符序列的压缩结果普遍好于其各自的算术编码压缩比(操作符序列的熵).

Table 1 The connectivity compression result of test triangular meshes (b/v)

表 1 测试用三角网格拓扑压缩结果(b/v)

Graph	#V	#F	TG	TG+	%TG	PA	PA+	%PA	EB*	EB+	%EB	JBS	JBS+	%JBS
Apple	867	1 704	0.81	0.41	49	0.75	0.20	73	2.17	1.30	32	1.24	0.99	20
Arrow	342	632	1.24	1.08	13	1.36	1.06	22	2.68	2.47	8	2.55	2.38	7
Delfin_low	212	420	3.25	3.09	4	2.23	2.19	2	3.80	3.21	16	3.66	3.55	3
Dice	2 253	4 502	1.88	1.66	12	1.60	1.41	12	2.66	2.36	11	3.24	3.04	6
Domino	561	804	1.44	1.10	24	1.65	1.29	22	1.94	1.61	17	3.31	2.48	25
Foot	2 154	4 204	1.91	1.77	7	1.46	1.32	10	2.94	2.60	12	2.34	2.22	5
Griffe	106	186	2.79	2.61	6	2.26	2.12	2	3.32	2.97	11	4.15	3.87	7
KotyouRan	10 237	17 094	1.31	1.30	1	1.18	0.75	36	2.42	2.07	14	3.01	2.84	6
MB1_r_leg	1 141	2 190	1.15	0.96	17	1.14	0.58	49	2.42	2.13	12	2.47	2.12	18
Monkeybr	1 022	1 984	1.45	0.97	33	1.29	0.82	36	2.89	1.87	35	2.88	1.99	31
Moon	1 656	3 264	0.92	0.56	39	0.92	0.45	51	2.25	1.84	18	1.62	1.43	12
Myhead	510	923	3.48	3.33	4	2.85	2.79	2	3.10	2.92	6	4.11	4.02	2
Output	14 039	20 870	1.42	0.98	31	1.41	0.63	55	2.11	1.57	26	2.64	1.97	25
Padlock	1 736	2 798	2.54	2.21	13	2.00	1.68	16	2.53	2.10	17	3.35	2.61	22
Poly_game	1 004	1 998	3.71	3.57	4	2.61	2.57	2	4.04	3.34	17	3.90	3.83	2
Rhino	229	454	3.53	3.16	10	2.45	2.39	2	4.10	3.43	16	4.33	4.18	1
RoseR	13 002	20 664	1.22	1.22	0	1.28	0.91	29	2.18	1.99	9	2.88	2.77	4
S_w29_44	313	568	3.09	2.97	4	2.71	2.44	9	3.65	3.15	14	5.04	4.34	14
Slonik	223	448	4.16	3.88	7	3.62	3.43	5	4.09	3.27	20	3.80	3.58	6
Table3	1 232	2218	1.94	1.56	20	1.95	1.25	36	2.43	2.17	11	2.90	2.12	27
Trashbin	4 284	5 594	1.67	1.61	4	1.45	1.32	9	1.95	1.79	8	3.09	2.49	19
Tw_goku	4 549	8 604	2.71	2.63	3	2.16	2.08	4	3.36	3.01	10	3.58	3.43	4
Weiner	14 673	29 037	0.91	0.92	0	0.61	0.53	13	2.47	1.77	28	1.67	1.59	5
Average					12.9			20.8			15.3			11.4

Table 2 The connectivity compression result of test non-triangular meshes (b/v)

表 2 测试用多边形网格拓扑压缩结果(b/v)

Graph	#V	#F	KA	KA+	%KA	KB	KB+	%KB
Apple4	867	849	0.79	0.56	29	0.79	0.78	1
Archelon	14 367	27 697	2.51	2.35	7	2.90	2.71	7
Asuka	4 795	8 997	3.28	2.77	16	3.48	2.90	16
Ballhead	5 706	8 986	3.14	1.62	48	3.05	2.12	30
Bear	12 489	22 633	2.24	1.85	17	2.88	2.28	21
Camel4	2 045	1 971	3.72	3.62	3	2.74	2.40	12
Cathead	14 324	28 110	0.77	0.64	17	2.31	1.30	43
Cessna	3 745	3 897	3.33	3.20	4	3.62	3.32	8
Chook	2 548	3 864	4.58	3.51	2	3.94	3.47	12
Desk	240	172	1.67	0.98	41	1.47	1.36	7
Dog4	1 871	1 454	3.18	3.14	1	2.13	1.95	8
Ejug	5 263	5 184	1.29	1.03	20	0.83	0.69	17
Kero	16 156	27 698	3.10	2.21	29	3.21	2.46	23
Kobun	704	830	4.18	3.88	7	3.43	3.25	5
Lara	8 260	7 393	2.92	2.83	3	2.27	1.94	15
Magnolia	806	1 247	3.51	3.08	12	3.58	3.23	10
MB1_hair	1 392	2 725	4.74	4.69	1	3.94	3.31	16
Picachu	5 181	9 291	3.01	2.17	28	2.70	2.34	13
Plate	8 318	15 999	3.77	3.67	3	3.61	3.08	15
Robotwars	300	508	3.65	2.53	3	3.77	3.16	16
Seitig_lila	178	192	2.79	2.54	9	3.10	2.77	11
Shuttle	310	393	3.46	3.04	12	3.63	3.50	4
Tiger	303	599	2.67	2.63	5	3.70	3.14	15
Vegeta	4 862	8 545	4.26	4.12	3	3.65	3.32	9
Wooddoll0	3 747	3 084	1.88	1.25	33	2.09	1.85	11
Average					14.1			13.8

4 结 论

本文主要针对网格拓扑压缩领域提出了一种新的、高效的、普遍适用的熵编码方法.该方法先计算网格遍历方法得到的操作符序列的 Huffman 编码,然后用基于上下文的方法编码这个 Huffman 结果.本文的熵编码方法与各自序列的熵值(大部分也是各自的网格拓扑压缩的最终压缩比)作比较,在绝大多数情况下都有所提高,部分模型的压缩比得提高得非常大.

本文的算法还有一些有待改进之处,特别是探索本文的熵编码方法是否对所有序列具有普遍性.

对于本文的熵编码方法,在压缩比方面还有改进的余地.比如,如果选择了更合适的上下文,其压缩比肯定会得到提高,甚至会得到很大的提高.

我们已发表的一篇论文(文献[20])中部分用到了本文的熵编码方法(压缩 KB 算法)的操作符序列,而本文针对这种熵编码作了普遍适用性研究,可得出结论,本文的熵编码方法普遍适用于网格拓扑压缩类算法的熵编码部分.

References:

[1] Touma C, Gotsman C. Triangle mesh compression. In: Proc. of the Graphics Interface. New York: ACM Press, 1998. 26−34.

[2] Deering M. Geometry Compression. In: Proc. of the Siggraph'95. New York: ACM Press, 1995. 13−20.

[3] Taubin G, Horn W, Lazarus F, Rossignac J. Geometry coding and VRML. Proc. of the IEEE, 1998,86(6):1228−1243.

[4] Alliez P, Gotsman C. Recent advances in compression of 3D meshes. In: Dodgson NA, Floater MS, Sabin MA, eds. Advances in Multiresolution for Geometric Modelling. Cambridge: Springer-Verlag, 2005. 3−26.

[5] Gotsman C, Gumhold S, Kobbelt L. Simplification and compression of 3D meshes. In: Iske A, Quak E, Floater MS, eds. Proc. of the Tutorials on Multiresolution in Geometric Modelling. New York: Springer-Verlag, 2002. 319−361.

[6] Peng JL, Kim CS, Kuo CCJ. Technologies for 3D mesh compression: A survey. ELSEVIER Journal of Visual Communication and Image Representation, 2005,16(6):688−733.

[7] Huffman DA. A method for the construction of minimum-redundancy codes. Proc. of the IRE, 1952,40(9):1098−1101.

[8] Witten IH, Neal RM, Cleary JG. Arithmetic coding for data compression. Communications of the ACM, 1987,30(6):520−540.

[9] Alliez P, Desbrun M. Valence-Driven connectivity encoding for 3D meshes. In: Magnenat-Thalmann N, Thalmann D, eds. Proc. of the Eurographics. Manchester: Springer-Verlag, 2001. 480−489.

[10] Khodakovsky A, Alliez P, Desbrun M, Schroeder P. Near optimal connectivity encoding of 2-manifold polygon meshes. Journal of the Graphic Models, 2002,64(3-4):147−168.

[11] Isenburg M, Snoeyink J. Face fixer: Compressing polygon meshes with properties. In: Proc. of the SIGGRAPH 2000. New Orleans, 2000. 263−270.

[12] Gumhold S, Strasser W. Real time compression of triangle mesh connectivity. In: Proc. of the SIGGRAPH'98. New York: ACM Press, 1998. 133−140.

[13] Isenburg M, Snoeyink J. Spirale reversi: Reverse decoding of Edgebreaker encoding. In: Proc. of the 12th Canadian Conf. on Computational Geometry. Fredericton, 2000. 247−256.

[14] Jong BS, Yang WH, Tseng JL, Lin TW. An efficient connectivity compression for triangular meshes. In: Proc. of the 4th Annual ACIS Int'l Conf. on Computer and Information Science (ICIS 2005). Washington: Publisher IEEE Computer Society, 2005. 583−588.

[15] Rossignac J, Lopes H, Safanova A, Tavares G, Szymczak A. Edgebreaker: A simple compression for surface with handles. Computers & Graphics Int'l Journal, 2003,27(4):553−567.

[16] Rossignac J. Edgebreaker: Connectivity compression for triangle meshes. IEEE Trans. on Visualization and Computer Graphics, 1999,5(1):47−61.

[17] Szymczak A, King D, Rossignac J. An Edgebreaker based efficient compression scheme for regular meshes. In: Proc. of the 12th Canadian Conf. on Computational Geometry. Fredericton, 2000. 53−68.

[18] Kronrod B, Gotsman C. Efficient coding of non-triangular mesh connectivity. Graphical Models, 2001,63(4):263−275.

[19] Lee H, Alliez P, Desbrun M. Angle-Analyzer: A triangle-quad mesh codec. Computer Graphics Forum, 2002,21(3):383−392.

[20] Liu Y, Wu EH. Connectivity compression for non-triangular meshes by context-based arithmetic coding. In: Proc. of the ACM SIGGRAPH GRAPHITE 2006 (the 4th Int'l Conf. on Computer Graphics and Interactive Techniques in Australasia and South-East Asia). Kuala Lumpur: ACM Press, 2006. 417−424.

刘迎(1970－),女,辽宁锦州人,博士,副研究员,主要研究领域为网格压缩.

孙春娟(1973－),女,博士生,副研究员,主要研究领域为计算机图形学网格压缩.

刘学慧(1968－),女,博士,副研究员,主要研究领域为计算机图形学.

吴恩华(1947－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为计算机图形学,科学可视化.