ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.12, December 2009, pp.3254−3264
doi: 10.3724/SP.J.1001.2009.03529

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

基于Wang Tiles的几何纹理合成*

韩建伟+, 王 青+, 周 昆, 鲍虎军

(浙江大学 CAD&CG 国家重点实验室,浙江 杭州 310058)

Wang Tile Based Geometric Texture Synthesis

HAN Jian-Wei+, WANG Qing+, ZHOU Kun, BAO Hu-Jun

(State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou 310058, China)

+ Corresponding author: E-mail: {hanjianwei, qwang}@cad.zju.edu.cn

Han JW, Wang Q, Zhou K, Bao HJ. Wang tile based geometric texture synthesis. *Journal of Software*, 2009,20(12): 3254−3262. http://www.jos.org.cn/1000-9825/3529.htm

Abstract: This paper presents geometric texture Wang Tiles to generate geometric textures on different object surfaces at run-time. A set of Wang Tiles is pre-computed from a specific geometric texture and used to generate geometric textures on different object surfaces. Although tile-based approaches are used for image textures, constructing geometric texture Wang Tiles presents additional challenges due to the differences between geometric textures and image textures. The geometric texture Wang Tiles is built automatically by using constrained synthesis to ensure geometric continuity across all possible combinations of tile arrangements. Once a tile set is built, it can be reused for texturing different output objects. This texturing algorithm consumes much less storage and computation than the existing methods.

Key words: texture synthesis; Wang Tiles; geometric texture; geometric modeling; geometric detail

摘 要: 提出了一种基于 Wang Tiles 的几何纹理合成方法来在不同物体表面上即时地生成几何纹理.首先根据给定的几何纹理预计算出一组 Wang Tiles,然后用这组 Wang Tiles 在不同的目标物体上即时生成新的几何纹理.尽管基于 Wang Tiles 的方法已经应用于图像纹理,但由于几何纹理采用了与图像纹理完全不同的表示方式,因此需要用完全不同的方法来处理.采用了基于约束的几何纹理合成技术自动生成几何纹理 Wang Tiles,从而保证了生成的几何纹理 Wang Tiles 在所有排列下都能保持其几何连续性.与现有的方法相比,生成的几何纹理 Wang Tiles 可以重用到不同的目标物体上,同时占用的存储空间及计算量更小,速度更快.

关键词: 纹理合成;Wang Tiles;几何纹理;几何建模;几何细节

中图法分类号: TP391 **文献标识码**: A

在图形学领域里,纹理映射是表示物体表面细节的一种有效方法.传统的纹理映射都是基于图像的,用图像纹理来表示物体的表面细节比直接的几何表示更为经济、有效,能够取得更快的绘制速度,并且占用更小的内存.但这种效率的提升是以绘制精度降低为代价的,图像纹理不支持遮挡、阴影、轮廓等重要的效果.随着显卡

* Supported by the National Natural Science Foundation of China under Grant No.60573152 (国家自然科学基金); the National Basic Research Program of China under Grant No.2009CB320800 (国家重点基础研究发展计划(973))

Received 2008-06-30; Accepted 2008-11-28

性能的提高,几何体的绘制已经不是当前的瓶颈,因此近几年人们提出了用几何纹理(geometric texture)代替图像纹理来表示物体的表面细节,从而提高绘制的精度.几何纹理比图像纹理表达力更强,可以解决图像纹理所面临的一系列问题.但是为物体表面添加几何纹理是一件极其复杂的工作,需要大量的手工交互才能取得满意的效果.

几何纹理和图像纹理一样都具有自相似性,但它比图像纹理更为复杂.几何纹理和图像纹理主要有以下差别:(1) 图像纹理是由规则排列的像素构成的离散表示;而几何纹理是由不规则拓扑连接的网格(mesh)构成的连续表示.(2) 图像纹理中的纹理元素(如纹理中的花)没有显式表示,因此很难提取,而几何纹理中的几何元素则可以通过拓扑连接关系提取出来.这些区别使得它们在合成时必须采用完全不同的方法.

几何纹理合成[1,2]是根据给定的样本几何纹理,通过合成的方法在物体表面上自动地生成新的几何纹理的技术,当前的几何纹理合成技术都是在目标物体表面上直接进行合成,它们虽然减少了添加几何纹理的工作量并且能够生成高质量的几何纹理,但合成的结果只能用于特定物体上,这样不仅降低了合成的速度,还需要大量的存储空间来保存每个物体上的合成结果.本文提出了几何纹理Wang Tiles(geometric texture Wang Tiles)技术.该技术首先从样本几何纹理中预生成一组Wang Tiles.我们可以用预生成的Wang Tiles在不同的物体表面上简单、快速地生成新的几何纹理.因此对于一个样本几何纹理,只要保存一组该纹理的Wang Tiles就可以了,这大极大地节省了存储空间.同时,采用几何纹理Wang Tiles只需通过简单的映射就可以在目标物体表面上生成新的几何纹理,加快了几何纹理的生成速度.

我们的方法主要源于图像纹理的Wang Tiles方法[3−5].为保证几何纹理Wang Tiles的边界连续性,使用了带约束的几何纹理合成技术根据样本纹理自动合成一组Wang Tiles.同时把物体进行基于Polycube的参数化[6],并根据Fu等人的方法[5]建立Wang Tiles与Polycube上四边形的对应关系,并通过壳映射技术(shell map)[7]把几何纹理映射到物体表面.

综上所述,本文的创新点主要有两点:一是提出了基于约束的几何纹理 Wang Tiles 生成方法.二是提出了基于 Polycube 的几何纹理 Wang Tiles 映射方法,大大加快了在物体表面生成几何纹理的速度.本文的方法可以广泛应用于几何建模,为现有建模工具提供表达力更强的几何纹理支持.

1 相关工作

1.1 基于样本的纹理合成

基于样本的表面纹理合成近年来取得了很大的进展.一类方法是基于逐像素非参数化采样[8,9]的表面纹理合成算法[10−12],这类方法对很多纹理并不适用.另一类方法是直接从样本纹理中拷贝纹理块进行纹理合成,这种方法适用于大部分的纹理.早期的方法是随机选取纹理块,并用alpha混合(alpha blending)来掩藏纹理块之间的接缝[13].文献[14–17]通过选择纹理块,使得块在接缝处更为连续,并通过alpha混合[16]或者最小割[14,15,17]来掩藏块之间的接缝.其他方法还有GPU上纹理合成算法[18,19]以及基于能量优化的方法[20].

1.2 Wang Tiles

Wang Tiles[3]是一组大小相同的可拼接正方形,这些正方形的每条边都给定一种颜色,相同颜色的边可以拼接起来,这些正方形称为Tile.一组Wang Tiles可以通过拼接来铺满整个平面,生成的图案是非周期性的.基于Efros等人[14]的方法,Cohen等人[3]提出了一种从样本纹理中自动生成Wang Tiles并用Wang Tiles来进行纹理合成的方法.为了提高Wang Tiles的生成质量,Ng等人[21]提出了ω-Tiles来减少Wang Tiles中的接缝.Wei[4]提出了一种Wang Tiles的排列组织方法来解决GPU上的随机访问问题,同时修正了纹理过滤问题.Fu等人[5]借助Polycube参数化[6]把Wang Tiles从平面推广到任意的物体表面.Leung等人[22]用Wang Tiles进行双向纹理函数(bidirectional texture function,简称BTF)的合成.但是这些算法只能处理像素表示的纹理(图像、高度场、BTF等),而不能处理具有一般拓扑结构的几何纹理.

1.3 几何纹理合成

几何纹理是指由几何元素构成的纹理.Bhat等人[1]使用了基于体素的方法在物体表面上合成实体纹理.Zhou等人[23]用样本数据来合成高度场表示的地形数据.Lagae等人[24]采用距离场的方法把网格转换成体素,然后通过体素合成来生成新的几何.但这些算法都不能处理更一般的网格模型.Merrell[25]首先把模型分解成片断,然后分析这些片断在样本中的排列方式,并通过重新排列这些片断生成新的模型.Zhou等人[2]提出了一种网格拼接(mesh quilting)技术,它把二维纹理的graph cut合成算法推广到三角形网格,根据样本模型生成新的模型.其合成的结果是一般的三角形网格,因此可以对生成的结果进行进一步的编辑.

2 算法概述

我们的算法基于几何纹理合成[2]和Wang Tiles[3,5].算法的输入是一个包围盒为$l_{in}\times w_{in}\times h_{in}$的样本几何纹理$M_{in}=\{V_{in},F_{in}\}$和基网格$M_{base}$,其中$M_{in}$和$M_{base}$都是三角形网格,而$V_{in}$是$M_{in}$的顶点集合,$F_{in}$是面集合.目标是在$M_{base}$上生成与$M_{in}$相似的几何纹理.首先通过$M_{in}$合成一组Wang Tiles,其中每个Tile都是一个包围盒为$w_T\times w_T\times h_{in}$的几何纹理,然后通过Wang Tiles拼接生成$M_{base}$上的几何纹理.

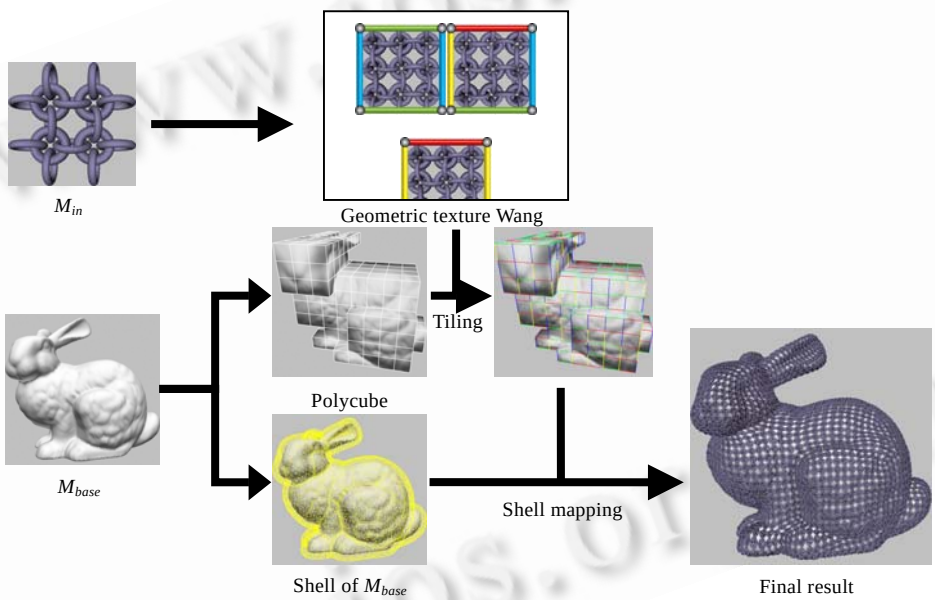


Fig.1 Pipeline of our algorithm

图 1 我们算法的流程图

我们采用Fu等人[5]的方法来保证生成的Wang Tiles能够映射到任意的物体表面,对基网格M_{base},用Polycube方法[6]进行参数化并对Polycube上每个四边形进行边着色.然后再根据边的颜色信息建立Wang Tiles与Polycube上四边形的对应关系.最后通过壳映射技术[7]把所有的几何纹理从纹理空间变换到壳空间,在M_{base}上生成新的几何纹理.图 1 是算法流程图.

3 几何纹理 Wang Tiles 的生成算法

现有的Wang Tiles生成算法[3,22]只能处理基于像素的纹理(如图像,BTF等).它们基于像素所构成的规则网格,用动态规划[14]或者最小割技术[17]进行合成.但是,几何纹理是由不规则拓扑连接的点面构成的,这导致这些算法并不适用.在合成Wang Tiles的同时还要考虑Tile之间可拼接边的连续性. Zhou等人[2]的方法虽然能合成几何纹理,但是不能够保证合成结果的边界满足用户的约束,因此我们在此基础上进一步添加了边约束来保证

Tile之间的几何连续性.我们首先根据边的颜色确定所需的边并把它们提取出来,然后用这些边构建边框并以这些边框作为约束来合成Tile的内部区域.

3.1 边提取

在对Wang Tiles进行拼接时首先要保证拼接的两条边的几何连续性,因此我们首先要生成Wang Tiles所需的边,再以它们为约束进行合成.从图 2 中我们可以看到,对Polycube而言只有相互平行的边才可以拼接,因此可以用同一组颜色对平行边进行着色.首先根据与X,Y,Z轴的平行关系把Polycube的边分成 3 类,对每一类都用同一组颜色进行着色.假设对平行于X,Y,Z轴的三类边着色所用颜色数分别为n_X,n_Y,n_Z,则所需边的总数就是$n_X+n_Y+n_Z$.

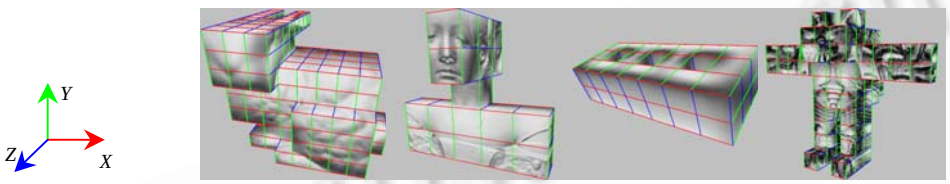


Fig.2 Classification of Polycube edges

图 2 Polycube 边的分类

确定所需边之后,我们根据Tile的大小$w_T\times w_T$以及用户给定边的高度h_E从M_{in}中选取得到这些边(如图 3 所示).不同类型的边(如X方向边和Y方向边)会在Tile的角上相互重叠.为了使重叠的区域能够无缝地连接起来,在选取边时要尽量保证这些边在重叠区域有对应的元素.重叠区域的元素并不要求完全吻合,后面我们会对元素进行变形融合来使它们更好地连接起来.当M_{in}太小时,我们可以先用Zhou等人[2]的方法从M_{in}生成一个较大的几何纹理M_{out},然后从M_{out}中选取所需要边.Cohen等人[3]提到,如果元素跨越Tile的角就会引入明显的瑕疵,因此要避免几何元素跨越角的情况.用户可以对提取的边进行编辑以取得更好的效果.

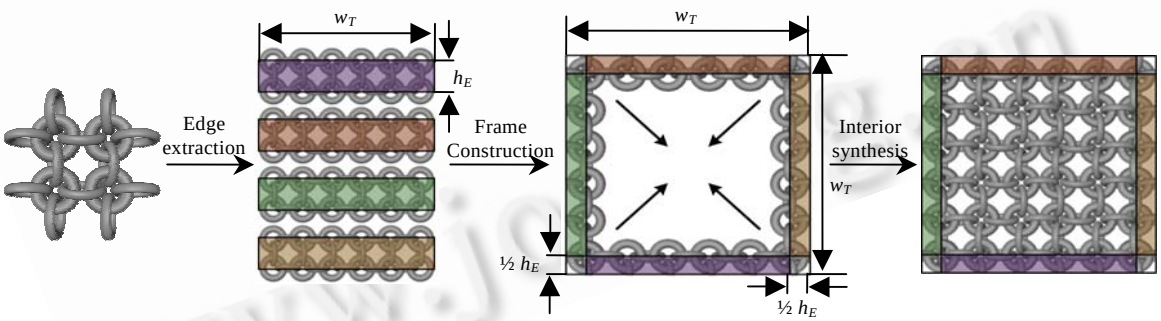


Fig.3 Pipeline of geometric texture Wang Tiles synthesis

图 3 几何纹理 Wang Tiles 的合成流程

3.2 边框构建

对于Wang Tiles中的每个Tile,我们根据其边颜色从提取出来的边中找到与之匹配的 4 条边(如图 3 所示),然后把这 4 条边放置到其对应的位置上构成当前Tile的框架.提取的每条边都是两个Tile共享的,因此只保留处于Tile内部的部分.在两条边的重叠区域,我们找出它们的对应元素,并根据Zhou等人[2]的方法对它们进行变形与融合.在变形和融合的过程中,Tile的边界必须保持不变以保证在拼接时能够保持几何上的连续性.在第 3.3 节将会描述如何在边界保持不变的情况下变形和融合对应的元素.

3.3 内部区域的合成

内部区域的合成是基于Zhou等人[2]的方法,其核心是对几何纹理中的元素(几何纹理中连接在一起的部分称为一个元素)进行拼接.我们对其进行了扩展来保证Tile中可拼接边的几何连续性.下面将详细描述合成其中一个Tile(记作M_T)的方法.

3.3.1 算法概述

内部区域的填充算法是带有边界约束的二维几何纹理合成.我们首先在*XY*平面上用大小相同的格子对M_{in}和M_T进行划分,同时记录M_T中每个格子是否被覆盖.初始状态M_T中被边覆盖的格子标识为“已占用”,其他格子标识为“未占用”.我们通过以下 5 个步骤来合成M_T的内部区域:(1) 寻找种子;(2) 几何匹配;(3) 找对应元素;(4) 元素变形;(5) 元素融合.

3.3.2 寻找种子

首先要找一个种子点作为合成的起点.为了能够不断地填充“未占用”的格子并与已合成部分保持一致,我们在“未占用”的格子中找一个其邻域中“已占用”格子最多的作为当前的种子点.

3.3.3 几何匹配

我们以种子点为中心在M_T上取一块比M_{in}小的区域P_T,然后把M_{in}放置到M_T中适当的位置,使它既能覆盖P_T,同时又与M_T中已合成部分最为匹配.首先使用Zhou等人[2]的方法找出M_{in}的近似最佳平移*t*,然后以这个平移为基础进行优化.

把M_{in}平移*t*之后的几何纹理记作$M_{in}(t)$,f^j_{in}是$M_{in}(t)$的一个面,则M_T的顶点v^i_T与f^j_{in}的距离定义为

$$D(v^i_T, f^j_{in}) = (1+\lambda Dist(v^i_T, f^j_{in}))(1+\left\|\boldsymbol{n}(v^i_T)-\boldsymbol{n}(f^j_{in})\right\|) \tag{1}$$

其中,$Dist(v^i_T, f^j_{in})$是v^i_T到f^j_{in}的距离,$\boldsymbol{n}(\cdot)$是法向,λ是权重.v^i_T到$M_{in}(t)$的距离是它到$M_{in}(t)$的所有面的距离中最小的:

$$E(v^i_T, M_{in}(t)) = \min_{f^j_{in}\in M_{in}(t)} D(v^i_T, f^j_{in}) \tag{2}$$

把距v^i_T最近的三角形记作f^i_{in},则该三角形上距离v^i_T最近的点h^i_{in}可以表示为

$$h^i_{in} = \alpha^i v^{i,1}_{in} + \beta^i v^{i,2}_{in} + \gamma^i v^{i,3}_{in} \tag{3}$$

其中($v^{i,1}_{in}, v^{i,2}_{in}, v^{i,3}_{in}$)是三角形$f^i_{in}$的 3 个顶点,($\alpha^i, \beta^i, \gamma^i$)是$h^i_{in}$相对于$f^i_{in}$的重心坐标.该平移下的匹配代价定义为

$$E(t) = \sum_{v^i_T\in P_T} E(v^i_T, M_{in}(t)) \tag{4}$$

Zhou等人[2]把平移*t*限制在格子上即只能移动格子大小的整数倍,因此能使M_{in}覆盖P_T平移是有限的,我们从这些满足条件的平移中找出能使公式(4)最小化的作为近似最佳平移*t*.

近似最佳平移 *t* 都被限制在格子上,因此可能并不是最优的(如图 4 所示).在我们的实验中发现,匹配结果的好坏与格子大小有很大的关系,对用户而言,选择适当的格子有很强的技巧性.因此我们在此基础上进行了进一步的优化,使得用户不必去关心格子的大小同时又能取得很好的效果.

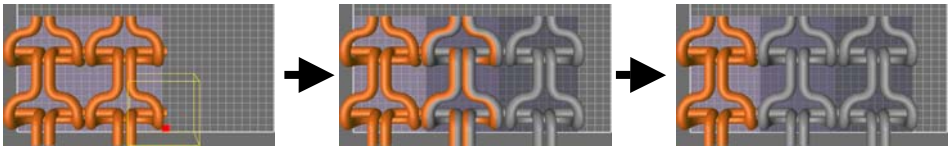
我们的优化基于迭代最近点(interactive closest points,简称ICP)[26]算法,该算法可以调整两个网格之间的位置,使得它们能够更好地匹配.基于近似最佳平移*t*,我们建立$M_{in}(t)$和M_T在重叠区域的对应关系,即对于M_T上处于P_T区域中的每个顶点v^i_T,在$M_{in}(t)$上找到它的最近点h^i_{in}.然后根据下面公式进行优化:

$$t \leftarrow t + \frac{1}{\#N(P_T)} \sum_{v^i_T\in P_T} (h^i_{in} - v^i_T) \tag{5}$$

其中,$\#N(P_T)$是P_T中的顶点数.我们根据式(5)不断优化*t*直到收敛,图 4 是我们的优化结果.

3.3.4 寻找元素对应关系

根据Zhou等人[2]的方法,对M_T中处于重叠区域的每个元素C_T,我们在$M_{in}(t)$中找到一组与之对应的元素.



(a) Initial output texture (a) 初始输出纹理 (b) Grid-Based matching (b) 基于网格的匹配 (c) After ICP optimization (c) ICP 优化之后

Fig.4 Geometry matching optimization

图 4 几何匹配优化

3.3.5 元素变形

对每个与输入元素C_{in}有关联的输出元素C_T,Zhou等人[2]用基于拉普拉斯的网格编辑技术[27,28]分别对它们进行变形,使它们既在几何上对齐又能保持局部几何细节.对C_T上处于重叠区域的顶点v_T^i,根据它在C_{in}上的对应点h_{in}^i,我们可以计算目标位置$c^i=(v_T^i+h_{in}^i)/2$.对于C_T而言,可以通过优化下面能量对其进行变形:

$$E_T(\{w^i\})=\sum_{i=1}^{N_T}\left\|L(w^i)-L(v_T^i)\right\|^2+\mu\sum_{i=1}^{m}\left\|w^i-c^i\right\|^2 \tag{6}$$

其中,$L(v)$是顶点v的拉普拉斯坐标,μ是权重.同样,对于C_{in}可以通过优化下面能量来得到最佳位置$\{w^i\}$:

$$E_{in}(\{w^i\})=\sum_{i=1}^{N_{in}}\left\|L(w^i)-L(v_{in}^i)\right\|^2+\mu\sum_{i=1}^{m}\left\|\alpha^i w^{i,1}+\beta^i w^{i,2}+\gamma^i w^{i,3}-c^i\right\| \tag{7}$$

其中,$(\alpha^i,\beta^i,\gamma^i)$是$h_{in}^i$的重心坐标.

元素只有处于重叠区域的部分发生了形变,但是重叠区域可能会跨越Tile的边,因此元素发生变形的部分仍然可能跨越Tile边界,这会导致Wang Tiles不具备可拼接性.因此,我们会找出元素中跨越边界的部分,在元素形变时保持这些部分不变.对新引入的元素C_{in}而言,要保证其最终融入M_T的部分不会跨越Tile的边界.

M_T的 4 条边分别记作b_0,b_1,b_2,b_3.对于其中的每个元素C_T,要找该元素跨越边界的部分,首先要确定它所属边的集合记作$B(C_T)$.在构建边框的时候若C_T被放在Tile的边b_i上并且与该边相交则$B(C_T)=\{b_i\}$,若C_T完全处于M_T内部则$B(C_T)=\varnothing$.在我们合成内部区域的时候会不断更新每个新生成元素C_T边的集合:若新元素完全处于M_T内部则$B(C_T)=\varnothing$,如果新元素是由两个元素C_1和C_2合并而来的,则新元素的边集合是C_1和C_2边集合的并集,即$B(C_T)=B(C_1)\cup B(C_2)$.对于输入元素C_{in},如果它在M_T中有对应的元素C_T则其边集合:$B(C_{in})=B(C_T)$,若没有对应元素则$B(C_{in})=\varnothing$.

如果元素中的三角形和元素边集合中任一条边相交,则称其为边界三角形,而该三角形的顶点就是边界点.在元素的变形过程中我们固定这些边界点来避免由于形变引起的 Wang Tiles 可拼接性的改变.

3.3.6 元素融合

完全处于M_T的元素不会改变Wang Tiles的边界,因此我们首先把那些完全处于M_T中的元素根据Zhou等人[2]的方法加入到M_T中.对于需要连接起来的元素C_T和C_{in},我们分别在两个元素上找一条路径使得两条路径最为靠近,然后沿着这两条路径把两个元素连接起来.元素上的路径可以用Graph cut算法[29]计算出来.以C_T为例,我们以它的三角形作为节点构建一个图,若两个三角形共用边是(v_T^i,v_T^j),则这两个三角形节点之间的权重为

$$(1+\left\|v_T^i-v_T^j\right\|)(1+Dist(v_T^i,C_{in})+Dist(v_T^j,C_{in})) \tag{8}$$

其中$Dist(v_T^i,C_{in})$是v_T^i到C_{in}的最短距离.我们在图上添加两个额外的节点:*SOURCE*表示要保留的三角形节点,*SINK*表示要删除的三角形节点.对于重叠区域之外的三角形以及边界三角形,我们用权重为无穷大的边连接到*SOURCE*节点.对于C_T上的顶点v_T^i,它在C_{in}上对应的三角形为f_{in}^i,若f_{in}^i是重叠区域之外的三角形或是边界三角形,或者f_{in}^i的邻接三角形和C_T中的任何顶点都没有对应关系,则与v_T^i相邻的三角形都用权重为无穷大的边连接到*SINK*节点.图构建好之后用graph cut算法把C_T分成两部分,与*SOURCE*相连的三角形节都保留下来,而与*SINK*相连的都删掉.对于C_{in},我们采用同样的方法得到它的路径.然后,用Yu等人[27]的方法沿着路径把两个元素融合起来.

经过上述步骤我们已经生成了所需的 Wang Tiles,并且能够把结果应用于平面上(如图 7 所示),在下一节将通过几何纹理映射来把得到的 Wang Tiles 映射到任意模型上.

4 几何纹理的映射

我们首先用Polycube[6]对基网格M_{base}进行参数化.Polycube与传统参数化方法的不同之处在于它的参数域是多个立方体组成的几何体的表面,是一组相接的正方形,因此可以把Wang Tiles映射到Polycube上.图 5 是一个二维的示意图,其中T_3是参数域(即Polycube表面). Polycube参数化的目标是把基网格M_{base}映射到T_3上,我们首先对M_{base}上的每个顶点v赋给一个唯一的三维纹理坐标 $v_{T_3} \in \mathrm{T}^3$,然后用一个投影函数$P:\mathrm{T}^3 \to T_3$把 v_{T_3} 映射到T_3上得到顶点v在Polycube上的对应点 f_{T_3}.

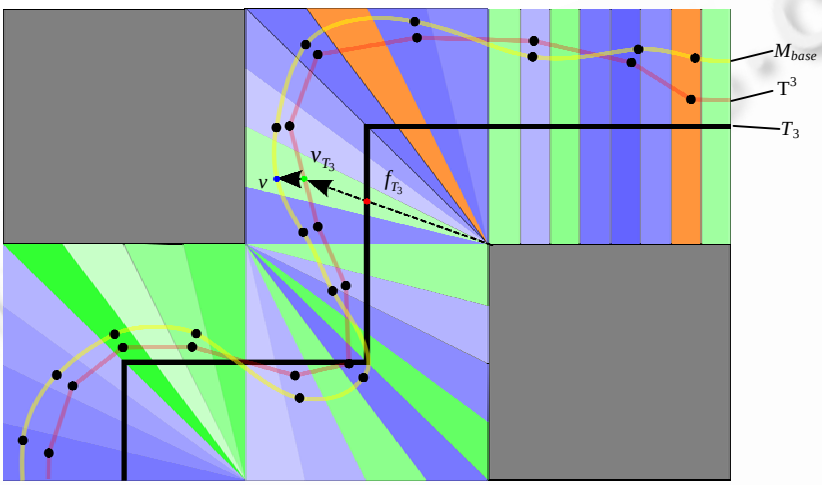


Fig.5 Shell mapping of geometric texture

图 5 几何纹理的壳映射

进行几何纹理映射之前,首先对Polycube上四边形的每条边进行着色,互相平行的边用同一组颜色(即生成Wang Tiles时所使用的颜色)进行随机着色.之后根据颜色对应找到Polycube上四边形与Wang Tile的对应关系.我们采用壳映射[7]把Wang Tiles中的几何纹理映射到物体表面,即从几何空间变换到壳空间.对于几何纹理Wang Tiles中的顶点v^T,根据它在Wang Tiles中的位置及Wang Tiles和Polycube的对应关系,我们可以计算出v^T在Polycube上的对应点 $f_{T_3} \in T_3$.然后通过一个反向投射$P^{-1}:\mathrm{T}^3 \to T_3$可以得到$v^T$在$M_{base}$上的对应点$v$的三维纹理坐标 $v_{T_3} \in \mathrm{T}^3$.因为顶点的三维纹理坐标和位置信息是一一对应的,因此我们可以进一步得到v^T在M_{base}上的对应点v.最后用壳映射把几何纹理变换到物体的表面从而得到最终的结果.

在计算反向投射$P^{-1}:\mathrm{T}^3 \to T_3$时,考虑到Polycube的立方体总共有 6 种形态,每种形态都采用不同的投射函数,因此我们首先判断要计算的顶点v^T在Polycube上的对应点 f_{T_3} 所在立方体的形态,然后分别采用相应的反向投射函数.为了快速计算反向投射方向与T^3的交点,我们对M_{base}的三维纹理域T^3建立了BSP树来进行加速.为了减少壳映射带来的扭曲,我们采用了低扭曲的外壳产生算法[2].图 5 是几何纹理映射的示意图.

5 实验结果分析

我们用 VC++2005 和 OpenGL 在 Windows Vista 下实现本文的算法,运行环境是 2.0G 的 Intel® Pentium® Dual CPU,2G 内存.图 6 是用我们的技术把 3 种几何纹理(从上到下分别是 Ring,Stone,Weave)分别合成到 3 个不同的基网格上(从左到右分别是 Holes,Bust,Bunny).

Fig.6 Results of geometric texture mapping. Left: Geometric textures. Right 3 columns: Three mapping results

图 6 几何纹理映射的结果.左边一栏是几何纹理.右边三栏是映射的结果

合成Wang Tiles时每个Tile的生成时间和Zhou等人[2]的方法相当,每个Tile一般需要几分钟时间.一组Wang Tiles所包含的Tile数和边颜色有关,如果平行于X,Y,Z轴的边所用的颜色数分别是n_X,n_Y,n_Z,那么构建一组完全的Wang Tiles就需要 $2\times(n_X^2n_Y^2+n_X^2n_Z^2+n_Y^2n_Z^2)$个Tile,对于大部分的几何纹理$X,Y,Z$轴分别使用两种颜色就可以生成满意的结果,这种情况下共需要 96 个Tile,而对于规则性很强的纹理(比如结果中的Ring),X,Y,Z轴各一种颜色就可以了,这种情况下只需要 6 个Tile.如果只把几何纹理应用到二维情况下(比如墙、地形等),则需要的Tile还会大为减少,这种情况下只需要 $2n_X^2n_Y^2$个Tile,即如果水平边和竖直边各两种颜色只要 16 个Tile就可以了.表 1 是我们的统计数据.

Table 1 Performance of geometric texture Wang Tiles synthesis

表 1 合成几何纹理 Wang Tiles 的性能

Geometric texture	M_{in}		M_T		Synthesis time (m)
	Vertices	Faces	Vertices	Faces	
Ring	4 608	9 216	7 920	15 552	0.5
Stone	6 176	12 288	24 906	49 152	1
Weave	7 704	15 168	7 803	15 158	2.5

几何纹理Wang Tiles合成之后,在模型上生成几何纹理的时间一般只需几秒钟,而Zhou等人[2]的方法则一般需要几十分钟甚至 1 个多小时.表 2 是几何纹理Wang Tiles映射到基网格上的性能统计.从中可以看到我们的算法在性能上有了很大的提高.

因为我们合成的是几何纹理,因此可以对结果进行进一步的处理.除了对结果进行编辑之外,还可以与基网格进行布尔运算,Hable等人[30]提供了一种GPU算法可以在绘制时直接进行布尔运算.生成的结果可以直接和基网格做布尔运算.图 7 是布尔运算的结果.

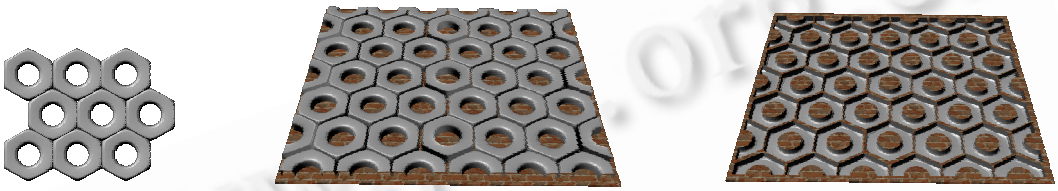
与其他表面纹理合成算法一样,我们的算法需要对模型进行较好的参数化,如果基网格的参数化扭曲严重,就不能得到很好的结果.图 8 是一个扭曲严重的例子,从中可以看到红框中的几何纹理由于参数化 的原因有较

大的扭曲.近年来,参数化技术的发展使我们能够挑选一个更好的参数化方法来避免这个问题.

Table 2 Performance of geometric texture Wang Tiles mapping

表 2 几何纹理 Wang Tiles 映射的性能

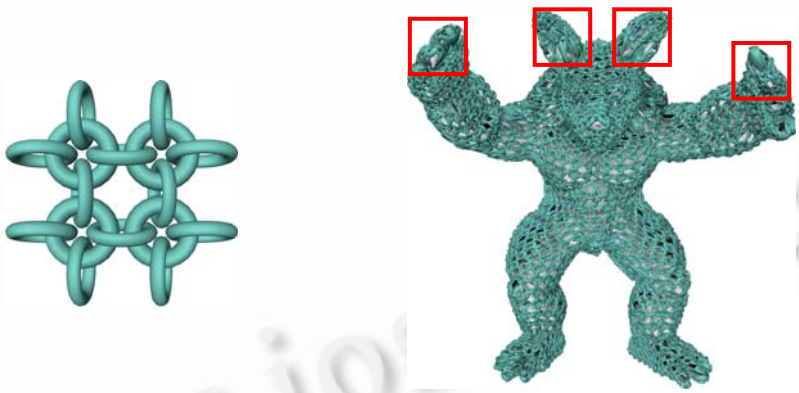
M_{Base}	Vertices	Faces	Number of tiles	Geometric texture	Time (s)
Holes	1 996	4 000	100	Ring	1.7
				Stone	5.4
				Weave	1.7
Bust	5 050	9 987	53	Ring	1.1
				Stone	3.6
				Weave	1.1
Bunny	7 502	15 000	322	Ring	7.9
				Stone	25.4
				Weave	7.9



(a) Geometric textures (a) 几何纹理 (b) Traditional mapping (b) 传统映射 (c) Boolean “difference” result (c) 布尔“差”结果

Fig.7 Boolean operation

图 7 布尔操作



(a) Geometric texture (a) 几何纹理 (b) Mapping result (b)映射结果

Fig.8 Distortion of geometric texture mapping

图 8 几何纹理映射的扭曲问题

从结果中可以看出,我们的算法比传统几何纹理合成算法无论在效率还是在存储空间上都具有很大的优势.几何纹理 Wang Tiles 使得纹理的合成和映射两部分完全分离,我们可以分别执行这两个部分,减少了模型对纹理的依赖性.这样的优势在于我们能够把同一组 Wang Tiles 应用到不同的物体上,增加了纹理的可重用性.同时,我们只需要合成二维的几何纹理并且映射方法简单,极大地降低了实现的难度.

6 结束语

本文提出了一种基于几何纹理 Wang Tiles 的几何纹理生成算法.与以往的算法不同,我们不是直接在基网格表面上进行纹理合成,而是通过样本纹理生成一组 Wang Tiles,然后根据这组 Wang Tiles 在不同的物体上生成

新的几何纹理.该方法不需要存储每个物体上的合成结果,因此大大减少了存储空间.同时又由于不需要对每个物体单独进行合成,而只是通过预生成好的 Wang Tiles 进行直接映射,从而加快了合成的速度.

我们的算法对于大部分几何纹理都能生成很好的结果.现在的结果都是针对各向同性的纹理的,在下一步的工作中我们会通过改进 Wang Tiles 算法来支持对各向异性几何纹理的合成.此外,现在的几何纹理映射方法是在 CPU 上完成的,我们会对其做进一步的改进,在 GPU 上实现实时映射.随着计算机性能的提高,人们对纹理的要求也进一步增加,纹理从图像发展到凹凸贴图(bump map),再到 BTF、置换贴图(displacement map)、壳纹理函数(STF)等,我们相信几何纹理作为一种更完善的纹理表示方式一定会有更广阔的应用.

References:

[1] Bhat P, Ingram S, Turk G. Geometric texture synthesis by example. In: Scopigno R, Zorin D, eds. Proc. of the 2004 Eurographics/ACM SIGGRAPH Symp. on Geometry Processing. New York: ACM Press, 2004. 41−44.

[2] Zhou K, Huang X, Wang X, Tong Y, Desbrun M, Guo B, Shum HY. Mesh quilting for geometric texture synthesis. ACM Trans. on Graphics (TOG), 2006,25(3):690−697.

[3] Cohen MF, Shade J, Hiller S, Deussen O. Wang Tiles for image and texture generation. ACM Trans. on Graphics (TOG), 2003,22(3):287−294.

[4] Wei LY. Tile-Based texture mapping on graphics hardware. In: Akenine-Möller T, McCool M, eds. Proc. of the ACM Siggraph/Eurographics Conf. on Graphics Hardware. New York: ACM Press, 2004. 55−63.

[5] Fu CW, Leung MK. Texture tiling on arbitrary topological surfaces using Wang Tiles. In: Bala K, Dutré P, eds. Proc. of the Eurographics Symp. on Rendering. 2005. 99−104.

[6] Tarini M, Hormann K, Cignoni P, Montani C. PolyCube-Maps. ACM Trans. on Graphics (TOG), 2004,23(3):853−860.

[7] Porumbescu SD, Budge B, Feng L, Joy KI. Shell maps. ACM Trans. on Graphics (TOG), 2005,24(3):626−633.

[8] Efros AA, Leung TK. Texture synthesis by non-parametric sampling. In: Proc. of the Int'l Conf. on Computer Vision. Washington: IEEE Computer Society, 1999. 1033−1038.

[9] Wei LY, Levoy M. Fast texture synthesis using tree-structured vector quantization. In: Brown JR, Akeley K, eds. Proc. of the 27th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 2000. 479−488.

[10] Turk G. Texture synthesis on surfaces. In: Pocock L, ed. Proc. of the 28th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 2001. 347−354.

[11] Wei LY, Levoy M. Texture synthesis over arbitrary manifold surfaces. In: Pocock L, ed. Proc. of the 28th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 2001. 355−360.

[12] Ying L, Hertzmann A, Biermann H, Zorin D. Texture and shape synthesis on surfaces. In: Gortler SJ, Myszkowski K, eds. Proc. of the 12th Eurographics Workshop on Rendering Techniques. London: Springer-Verlag, 2001. 301−312.

[13] Praun E, Finkelstein A, Hoppe H. Lapped textures. In: Brown JR, Akeley K, eds. Proc. of the 27th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 2000. 465−470.

[14] Efros AA, Freeman WT. Image quilting for texture synthesis and transfer. In: Pocock L, ed. Proc. of the 28th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 2001. 341−346.

[15] Soler C, Cani MP, Angelidis A. Hierarchical pattern mapping. ACM Trans. on Graphics (TOG), 2002,21(3):673−680.

[16] Liang L, Liu C, Xu YQ, Guo B, Shum HY. Real-Time texture synthesis by patch-based sampling. ACM Trans. on Graphics (TOG), 2001,20(3):127−150.

[17] Kwatra V, Schödl A, Essa I, Turk G, Bobick A. Graphcut textures: Image and video synthesis using graph cuts. ACM Trans. on Graphics (TOG), 2003,22(3):277−286.

[18] Lefebvre S, Hoppe H. Parallel controllable texture synthesis. ACM Trans. on Graphics (TOG), 2005,24(3):777−786.

[19] Lefebvre S, Hoppe H. Appearance-Space texture synthesis. ACM Trans. on Graphics (TOG), 2006,25(3):541−548.

[20] Kwatra V, Essa I, Bobick A, Kwatra N. Texture optimization for example-based synthesis. ACM Trans. on Graphics (TOG), 2005,24(3):795−802.

[21] Ng TY, Wen C, Tan TS, Zhang X, Kim YJ. Generating an ω-tile set for texture synthesis. In: Guo B, Pfister H, Samaras D, eds. Proc. of Computer Graphics Int'l 2005. Washington: IEEE Computer Society, 2005. 177−184.

[22] Leung MK, Pang WM, Fu CW, Wong TT, Heng PA. Tileable BTF. IEEE Trans on Visualization and Computer Graphics, 2007,13(5):953−965.

[23] Zhou H, Sun J, Turk G, Rehg JM. Terrain synthesis from digital elevation models. IEEE Trans. on Visualization and Computer Graphics, 2007,13(4):834−848.

[24] Lagae A, Dumont O, Dutre P. Geometry synthesis by example. In: Spagnuolo M, Belyaev A, Suzuki H, eds. Proc. of the Int'l Conf. on Shape Modeling and Applications 2005. Washington: IEEE Computer Society, 2005. 176−185.

[25] Merrell P. Example-Based model synthesis. In: Spencer SN, ed. Proc. of the 2007 Symp. on Interactive 3D Graphics and Games. New York: ACM Press, 2007. 105−112.

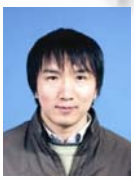
[26] Besl PJ, McKay ND. A method for registration of 3-D shapes. IEEE Trans. on Pattern Analysis and Machine Intelligence, 1992,14(2):239−256.

[27] Yu Y, Zhou K, Xu D, Shi X, Bao H, Guo B, Shum HY. Mesh editing with poisson-based gradient field manipulation. ACM Trans. on Graphics (TOG), 2004,23(3):644−651.

[28] Sorkine O, Cohen-Or D, Lipman Y, Alexa M, Rössl C, Seidel HP. Laplacian surface editing. In: Boissonnat JD, Alliez P, eds. Proc. of the 2004 Eurographics/ACM SIGGRAPH Symp. on Geometry Processing. New York: ACM Press, 2004. 175−184.

[29] Boykov Y, Veksler O, Zabih R. Fast approximate energy minimization via graph cuts. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2001,23(11):1222−1239.

[30] Hable J, Rossignac J. Blister: GPU-Based rendering of Boolean combinations of free-form triangulated shapes. ACM Trans. on Graphics (TOG), 2005,24(3):1024−1031.



韩建伟(1981－),男,河南开封人,博士,主要研究领域为纹理合成.

周昆(1977－),男,博士,教授,博士生导师,主要研究领域为几何处理,纹理合成,实时绘制.

王青(1976－),男,博士,讲师,主要研究领域为计算机图形,计算机辅助几何设计,数字几何处理.

鲍虎军(1966－),男,博士,研究员, 博士生导师,CCF 高级会员,主要研究领域为真实感图形学绘制技术,虚拟现实.