条纹理映射*

李奎宇[1,2+], 王文成[1], 吴恩华[1,3]

[1](中国科学院 软件研究所,计算机科学重点实验室,北京 100080)

[2](中国科学院 研究生院,北京 100039)

[3](澳门大学 科技学院计算机与信息科学系,澳门)

Bar Texture Mapping

LI Kui-Yu[1,2+], WANG Wen-Cheng[1], WU En-Hua[1,3]

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

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100039, China)

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

+ Corresponding author: Phn: +86-10-62522028, Fax: +86-10-62563894, E-mail: lky@ios.ac.cn, http://www.ios.ac.cn

Received 2004-04-05; Accepted 2004-07-05

Li KY, Wang WC, Wu EH. Bar texture mapping. *Journal of Software*, 2004,15(Suppl.):179~189.

Abstract: An image-based rendering method that can successfully support the representation of 3-D surface details and view motion parallax is presented. Expensive task of hole-filling is finished beforehand in the preprocessing stage, only texture-mapping hardware is employed in the rendering stage for acceleration. For correct representation, each pixel in a parallel projection image with depth is decomposed into several consecutive layers so that a bar texture is generated for that pixel. Since all the bar textures as a whole constitute a continuous 3-D approximation to the scene represented in the original depth image, novel views can be obtained by mapping these bar textures in a from-back-to-front order. Experimental results show the new approach can produce high visual quality and runs faster than many existing image-based rendering techniques with only moderate storage overhead.

Key words: bar texture; texture mapping; image-based rendering; acceleration; hardware

摘 要: 提出一种新的基于图像绘制的方法,能高效地利用图形硬件进行加速,避免成像过程中烦人的空洞填补计算,高质量地反映物体表面的三维凹凸细节及视差变化.该方法首先为源深度图像中每个像素生成一个条状的纹理(条纹理),即根据一个像素及其邻近像素的深度值,沿着深度方向为该像素插值生成多个具有不同深度值的点,它们一起构成该像素的条纹理;然后,绘制时利用图形硬件的纹理映射直接处理这些条纹理.由于各个像素的条纹理的集合可以构成源场景的近似连续三维表示,从而大大增强了源深度图像表达三维模型细节的能力,而且成像时不必

* Supported by the National Natural Science Foundation of China under Grant Nos.60373051, 60033010, 60173022 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312102 (国家重点基础研究发展规划(973)); the Research Grant of University of Macao of China (澳门大学基金资助项目)

作者简介: 李奎宇(1980－),男,山东莒县人,硕士,主要研究领域为计算机图形学,基于图像的绘制;王文成(1967－),男,博士,研究员,博士生导师,主要研究领域为计算机图形学,科学计算可视化,虚拟现实;吴恩华(1947－),男,博士,研究员,博士生导师,主要研究领域为计算机图形学,科学计算可视化,虚拟现实.

进行空洞填补的计算.与已有的同类方法相比,新方法的空间需求小,成像速度快,并且成像质量很高.

关键词: 条纹理;纹理映射;基于图像的绘制;加速;图形硬件

纹理长久以来就被用来模拟景物表面的微观细节,使计算机生成的虚拟场景更为真实、自然.但传统的纹理映射方法难以表现物体表面的三维凹凸细节.解决这一问题的方法通常有 3 种:凹凸纹理映射(bump mapping)[1],位移映射(displacement mapping)[2,3]和体纹理(volumetric texture)[4,5].凹凸纹理映射通过扰动表面法向来表达三维细节,但其前提是物体表面起伏很小,不会改变纹理映射后的轮廓线,所以它无法处理凹凸表面在视点改变后所引起的自遮挡现象.位移映射为每一个纹理像素赋予一个位移值,并将每一个像素看作一个多边形网格,计算量过大.体纹理映射虽然得到了硬件支持,但它需要巨大的存储量和计算量,且相关硬件价格昂贵,在目前无法普及.

基于图像的绘制技术(image-based rendering)是近年来图形学界的一个研究热点.它无须构造三维几何模型及进行复杂的光照运算就可生成具有高度真实感的新视点图像,且绘制速度只与图像的大小与分辨率有关.由于三维场景在目标图像上的投影与场景中各可见点的空间位置有关,因此,视点改变时,场景中各部分间的遮挡关系会改变,且它们成像时的位移也不同.此时,用简单的二维投影变换[6]将源图像像素变换到目标图像上是不正确的.解决这一问题的有效办法是在源图像中提供可见点的深度信息,以三维变换(three-dimensional image warping)[7]的方式将源深度图像中的像素变换到目标图像的适当位置,以保证成像的正确性.

作为三维变换的一种优化处理,浮雕纹理映射(relief texture mapping)[8]能高效地利用图形硬件的纹理映射进行加速,同时高质量地反映物体表面的三维凹凸细节及视差变化.但该方法在进行纹理映射前要有一个前处理,以生成满足纹理映射要求的中间图像(pre-warped texture)[8],且该前处理是用软件计算的方式完成,它占用了成像过程 90%以上的时间[8],极大地影响了效率.基于硬件加速的浮雕纹理映射(hardware-assisted relief texture mapping)[9]利用图形卡的像素可编程性,通过使用多重纹理映射(multi-texture mapping)省去了中间图像的生成过程,而直接成像.该方法虽然提高了速度,但其简单的空洞填补策略使得成像粗糙且不完整.层次纹理映射(layered texture mapping)[10]提出将深度图像的各个像素进行分层处理,并将空洞填补的操作前移到生成层次纹理的预处理中.因此,成像时可以直接使用纹理映射,且不必进行空洞填补的计算,很好地提高了速度,并能高质量地反映物体表面的三维凹凸细节及视差.但其预处理得到的层次纹理数据对外存储空间要求较大.

在此,本文提出一种条纹理映射的方法,很好地处理了前面算法中存在的种种不足.它把深度图像转化为一组条纹理(即根据一个像素及其邻近像素的深度值,沿着深度方向为该像素插值生成多个具有不同深度值的点,它们一起构成该像素的条纹理),生成了场景的一种近似连续的三维表达.这样,成像时只需利用图形硬件来处理这些条纹理就可快速生成目标图像,而不必进行空洞填补.并且新方法生成的图像能够高质量地反映场景的三维细节及视差变化.

在运用条纹理成像时,本文采用两种条纹理映射策略:条纹理的选择映射和全映射.前者仍然按照浮雕纹理映射的流程进行成像,只是利用纹理映射的硬件,通过有选择地映射那些可见的条纹理,取代了浮雕纹理映射中软件计算填补空洞的过程,从而大大节省了前处理的时间.并且,随着可编程图形硬件性能的发展,生成中间图像的前处理,包括用于空洞填补的条纹理映射,都可以完全在 Pixel Buffer[11,12]里完成,由此可以进一步提高效率.而条纹理的全映射根据 McMillan 提出的极线原则[7,8],按照由远到近的顺序依次映射源图像的每一个条纹理来完成成像,而不需要进行深度比较就能解决遮挡的判定.新方法绝大部分计算依靠纹理映射完成,因此能够方便地使用图形硬件进行加速.

对于带有多层几何特征的复杂场景,为了避免由可见性变化而产生的空洞,我们把条纹理和多层深度图像(layered depth images)[6]的思想相结合,进一步提出了多层条纹理(layered depth bars)的模型表示和绘制的方法,以正确绘制那些层次复杂的场景.

我们在统一的测试平台上,分别实现了新方法和几种相关的算法,以进行实验对比.结果显示,本文提出的条纹理映射在综合考虑成像质量、速度和存储空间的要求下,比其他几种同类算法的效率要高很多.

本文第 1 节介绍相关的工作.第 2 节详细介绍本文提出的条纹理映射方法.第 3 节讨论实验结果.第 4 节进

行总结.

1 相关工作

基于图像绘制里的三维变换[7]描述了空间点在不同图像平面上的对应像素间的变换关系.它根据带有深度信息的源图像 i_s 生成关于新视点的目标图像 i_t 源图像所表达场景的三维几何信息是通过每个可见点的深度和此图像的针孔相机模型(pin-hole camera)[7]来表示的.假设空间中一点 $X(x,y,z)$ 在源图像平面 i_s 上的投影点 x_s 的坐标是 (u_s,v_s),则这一点在目标图像 i_t 上的对应点可以表示为

$$x_t = \delta(x_s')P_t^{-1}(C_s - C_t) + P_t^{-1}P_s x_s \tag{1}$$

这里,P_s 和 P_t 分别是图像 i_s 和 i_t 的相机矩阵(pinhole-camera matrix)[7],$x_s=[u_s,v_s,1]^T$,C_s 和 C_t 分别是图像 i_s 和 i_t 的针孔相机模型的投影中心(center of projection)[7],$\delta(x_s)$ 是点 x_s 的普遍视差(generalized disparity)[7].由公式(1)可知,将源图像进行二维投影变换后,所得中间图像上每个像素再沿着朝向目标图像的极点(epipole)[7]方向做一个正比于此像素普遍视差 δ 的位移,即可得到新视点图像.

浮雕纹理映射[8]将三维变换的公式拆分成对源图像进行两次分别沿行与列的一维变换和随后的二维纹理映射两步[13,14],以便二维纹理映射可以利用硬件加速.作为该算法的前处理,两次一维变换把源图像变换成满足第 2 步纹理映射要求的中间图像.其思想是:对源图像上的每一个点 (u_s,v_s),使得它在中间图像上的对应点 (u_i,v_i) 经二维纹理映射变换到目标图像上的点,与 (u_s,v_s) 经三维变换到目标图像上的点一致.在此,中间图像可以通过对源图像上的每个像素做两次在行与列上的位移获得,位移量与像素的深度和视点相关.该方法能高效地反映三维凸凹细节和视差变化,并且基于纹理映射的成像速度很快.但其前处理要进行空洞填补的计算,在整个绘制过程中占用了 90%以上的时间[8].

由于浮雕纹理映射中的大部分时间花费在前处理生成中间图像上,硬件加速的浮雕纹理映射[9]将原算法的前处理工作交给最新图形硬件支持的像素着色引擎(pixel shader)去完成.根据当前视点位置,它首先生成关于源图像的每个点到它在中间图像上对应点的位置变化的位移图(offset map)[9].绘制时,通过使用多重纹理映射,直接完成成像.但是,对于目标图像上两个原先相邻像素的分离而产生的空洞,它只能通过填补位移图上对应的空洞来进行消除,而在填补位移图上的空洞时,它只是将前一个像素点的位移值直接赋给空洞处像素.而且,它不能从场景包围盒多个面对应的多幅参考图像完整地合成目标图像.所以,虽然该方法的显示速度很快,但其成像粗糙且不完整.不同于这些浮雕纹理映射的方法,本文提出的条纹理映射通过预处理的条纹理生成,既能进行高质量的插值,又不必在成像时进行空洞填补的计算.且绘制时只需利用图形硬件处理这些条纹理,而不是单个的像素.因此,新方法的成像速度快,且成像质量很高.

层次纹理映射[10]是另一种高效的基于图像绘制的算法.该算法将带有深度信息的图像的像素按照深度分层存放,得到具有不同深度值的层次纹理(layered textures)[10],以便成像时使用纹理映射的硬件将这些层次纹理由后往前依次映射到成像面上,而且能方便地表达物体表面三维凹凸细节和视差的变化.为了避免目标图像上出现空洞以节省绘制时填补空洞的计算,该方法在预处理生成层次纹理时,将每个像素在几个深度层次上进行扩展.对于场景包围盒各个表面所对应的深度图像,都分别生成其层次纹理,这样就形成了对整个三维场景的层次纹理表示.由于成像时无需进行空洞填补而只用纹理映射完成绘制,该方法无论在成像速度和质量上都要比浮雕纹理映射优越.但该算法的层次纹理数据需要较大的存储空间,不便于处理大型复杂的模型.本文的条纹理映射与层次纹理映射的一个相似点是,都需要为源深度图像上的每个像素在深度方向上进行扩展.但是,条纹理的表示方法所需要的外存储空间比层次纹理要小得多,且便于利用图形硬件进行绘制.

2 条纹理映射

传统的二维纹理映射难以表达物体表面的三维凹凸细节和视差变化.为了实现这一点,并且在绘制时能够应用图形硬件进行加速,我们把一幅带有正投影深度(orthogonal displacement)的纹理图像分解成一组条纹理的集合来表示.这一组条纹理将构成源深度图像中场景的近似连续三维表示,由此可以省去成像时的空洞填补操

作(如图 1 所示).

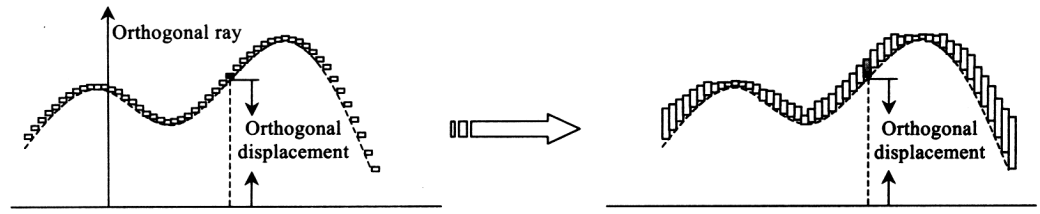

图 1 条纹理增强了源深度图像的表达三维几何信息的能力

基于深度图像的条纹理表示,本文在第 2.2 节提出两种条纹理映射策略:条纹理的选择映射和全映射.前者实际上是对浮雕纹理映射的一个高效加速方法.它用对条纹理的映射取代了原算法中软件计算填补空洞的过程,从而把原先前处理阶段绝大部分在 CPU 上的计算改用图形硬件的纹理映射来实现,大大缩短了生成中间图像的时间.而条纹理全映射方法则是采用极线原则[7,8],按照由远到近的顺序依次映射这些条纹理来完成成像.极线原则保证了成像时正确的遮挡关系,且不需要进行深度比较.由于这些绘制方法中的大部分计算是基于纹理的映射,因此新方法能很好地运用图形硬件进行加速.而对具有多层几何结构的复杂场景,我们在第 2.3 节提出多层条纹理的方法就能方便地进行处理.

2.1 条纹理的生成

简单地说,条纹理是二维深度图像上为每个像素在深度方向上扩展的结果.对深度图像上每一个像素,根据它和相邻像素的深度,沿其深度方向在几个连续的深度层次上为该像素插值生成几个具有不同深度值的点,这些插值得到的点和源像素一起组成了该像素的条纹理.而对于没有被扩展的像素,该像素本身将构成一个条纹理.如图 2 所示的二维示意图,对像素 B 插值所得到的点与 B 一起构成了 B 处的条纹理.

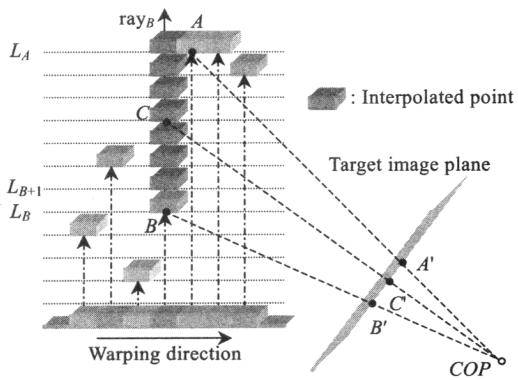

图 2 B 点处的条纹理将用来填补成像面上 A' 与 B' 之间的空洞

显然,源深度图像中所有像素的条纹理一起能构成源场景的一种近似连续的表达.这样,绘制时通过处理条纹理而不是像素,就能有效地填补空洞.如图 2 所示,当源图像的各个像素按照其深度值位于三维空间相关的位置时,从视点 COP 观察这些点,有些像素比如 A 和 B,它们在目标图像上的投影 A' 与 B' 之间就会出现空洞.显然,如果是映射 B 像素的条纹理,这些空洞将被自动填补.

根据文献[10]的讨论,对一个像素进行深度方向的扩展时,总是往深度值比该像素的深度值大的方向扩展,且扩展的长度是由该像素的邻近像素中与它属于同一个连续面且深度比它大的像素的深度决定的.如图 2 所示,对于空间中属于同一个连续面但深度差比较大的两相邻像素 A 和 B,对深度值较小的像素 B,需要沿深度方向 ray_B,从 L_{B+1} 到 L_A 之间的每一层的同一位置,都插值生成一个新的点.

在真正的二维深度图像中,一个像素的扩展结果与它周围 8 个相邻像素相关,其扩展长度由相邻像素的最大深度差决定,而各个插值点的颜色是所有对它有影响的像素颜色的加权平均[10].根据上一段的讨论,一个像素是不会对与它相邻且深度比它大的插值点的颜色有影响的.所以,对某一插值点的颜色,只有深度值大于该点的相邻像素才会对其有影响.如图 3 所示的三维示意图,这是深度图像中相邻的 9 个像素按其深度放置的情况,中间像素 O 在其深度方向上的 6 个不同的深度层次上进行了扩展,由近往远增加了 O_1,O_2,O_3,O_4,O_5 和 O_6 这 6 个

新的点,每个扩展的点 $O_i(i=1,\ldots,6)$的颜色为 A~H 中所有深度层次大于 $O_i(i=1,\ldots,6)$的像素与像素 O 颜色的加权平均.例如,由于像素 A,B 的深度层次在 O_1~O_6 之前,所以它们对所有插值点的颜色无贡献.依此类推,O_1 和 O_2 的颜色为像素 C~H 和 O 的颜色的加权平均,O_3 的颜色为像素 D~G 和 O 的颜色的加权平均,O_4 的颜色是像素 D~F 和 O 的颜色的加权平均,对 O_5 和 O_6 的颜色有影响的像素是 E,F 和 O.所有得到的插值点 O_1~O_6 与 O 一起构成了像素 O 的条纹理.

如果深度图像上两相邻像素是深度非连续的,即这两个相邻像素分别位于两个不同的连续面上.那么,对它们在成像面上的投影之间的"空洞"进行填补会引起过多填补(skins)[8]的错误.因为生成条纹理是为了保证场景中连续的部分在成像时没有空洞,因此,对于不连续的相邻部分,则没有必要插值生成条纹理.为此,如同已有的工作[8~10],本文的算法也通过设置一个深度差的阈值来判断相邻像素是深度连续的还是非连续的,且非连续的像素将不参加条纹理的生成.由于条纹理的长度是由相邻的、深度连续的像素的深度差决定的,所以它一般不会超过该阈值.这保证了所有条纹理都不会太长,所需的存储空间也就不会太大.

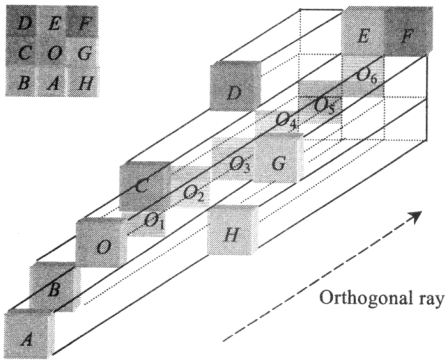

图 3 条纹理:深度图像中像素的扩展

与浮雕纹理映射[8]类似,本文对一个三维场景完整的模型表示是用其包围盒所对应的 6 幅深度图像的条纹理来表达的.

2.2 条纹理的绘制

在生成了源深度图像的条纹理之后,就可以利用图形硬件处理这些条纹理,以生成目标图像.根据第 2.1 节的讨论,基于条纹理的成像不必考虑空洞填补的问题,本节主要讨论如何进行条纹理的绘制.在此,我们采用两种条纹理映射方法:条纹理的选择映射和条纹理的全映射.前者基本上是按照浮雕纹理映射[8]的流程,只对那些用来填补空洞的条纹理进行映射,以取代原来在 CPU 上的填补空洞计算,由此加速了中间图像的生成.后者主要是基于极线原则[7,8]由远到近地逐个映射条纹理以进行成像,不必进行深度比较,而且可以运用比较简单的硬件功能进行成像.

2.2.1 条纹理的选择映射

由前面的分析可知,若源图像中位于同一连续面上的两个相邻像素深度差过大,它们在目标图像上的投影之间会有空洞出现.对此,将其中远离视点的像素的条纹理映射到成像面上就能填补这种空洞,因为由第 2.1 节的讨论可知,该空洞对应的就是远离视点的像素的条纹理.如图 2 所示,对于成像平面上 A' 与 B' 之间的空洞,只要把远离视点 COP 的像素 B 的条纹理映射到成像面上 A' 与 B' 之间,就可以正确完成空洞填补.

基于这种考虑,条纹理的选择映射改进了原浮雕纹理映射生成中间图像的过程.具体地,这是分两步完成的:

- 首先,采用浮雕纹理映射算法生成一幅中间图像[8],但在这一步里不必进行空洞填补的插值计算;
- 然后,对源图像上的每一个像素,判断该像素和与其相邻并且靠近极点[7]方向的 3 个像素在中间图像上的对应点是否会发生分离.若是,就可以把该像素的条纹理映射到第 1 步得到的中间图像上,以填补相应的空洞.此时,条纹理的映射只要根据视点方位运用纹理映射的硬件即可完成.

把两步的绘制结果"叠加"之后就可以获得完整的中间图像.最后,将得到的中间图像进行到空间多边形的二维纹理映射,就可以获取新视点图像[8].

如图 4 所示,源图像上与 A 相邻并靠近极点方向的 B,C,D 三点在中间图像上的对应点 B',C',D' 都会与 A' 分离.这就需要把像素 A 的条纹理分别映射在 A' 与 B'、A' 与 C' 和 A' 与 D' 之间,以填补相应的空洞.而对 B 和与其相邻并靠近极点方向的 D,E,F 三点在中间图像上的对应点中,只有 B' 与 D' 会发生分离,这时只需把像素 B 的条纹理映射在 B' 与 D' 之间即可.

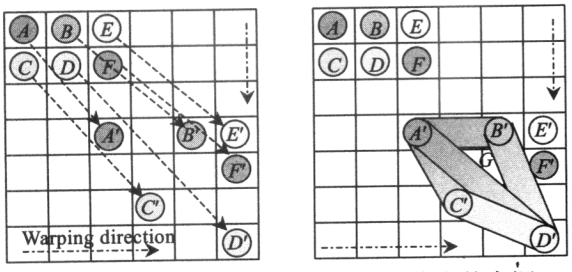

图 4 用条纹理的映射填补中间图像上的空洞

在实际操作中,条纹理是有宽度的.所以,对于如图 4 所示用条纹理的选择映射填补空洞之后仍然存在的"缝隙"G,是可以通过控制条纹理的宽度来覆盖的.实验证明,对于表面起伏不大的模型,使用一个像素宽的条纹理就可以恰好填补所有可能出现的空洞.

目前,利用最新可编程图形硬件的 Pixel Buffer[11,12]和渲染至纹理(render to texture)[15]功能,能够快速地进行中间图像(intermediate image)和动态纹理(dynamic texture image)的生成,并可以直接将绘制的结果送入纹理内存,而不受当前显示属性的限制.基于这一功能,中间图像的生成,包括用来填补空洞的条纹理的选择映射,都可以完全在 Pixel Buffer 里完成,以加速原浮雕纹理映射的成像.图 5 给出了条纹理的选择映射加速浮雕纹理映射的伪代码.

```
BarTextures_Assisted_Relief_Texture_Mapping( ) {
  //利用 Pixel Buffer 和 Render_to_Texture 生成中间图像
  {
    //以下的绘制过程全部在 p-buffer 里进行
    render_to_texture.BeginCapture(TexID);
      //首先获得未经空洞填补的中间图像
      Compute_Unholefilled_Prewarped_Texture( );
      //在 p-buffer 里绘制第 1 步得到的中间图像
      Render_Unholefilled_Prewarped_Texture( );
      //用条纹理选择映射填补空洞
      Bar_Texture_Mapping( );
    render_to_texture.EndCapture( TexID );
  }
  //至此已经利用 Render_to_Texture 获得完整的中间图像
  Mapping_to_ViewPlane( TexID );
}
```

```
Bar_Texture_Mapping( ) {
  for( y=ystart; y!=yend; y+=ystep )
    for( x=xstart; x!=xend; x+=xstep ) {
      //依次判断(x,y)和与其相邻且靠近极点方向的 3 个
      点在中间图像上的对应点是否会发生分离.若是,则
      要通过映射(x,y)的条纹理来填补空洞
      One_Bar_Mapping( x, y, x + xstep, y );
      One_Bar_Mapping( x, y, x, y + ystep );
      One_Bar_Mapping( x, y, x + xstep, y + ystep );
    }
}
```

```
One_Bar_Mapping( x, y, x1, y1 ) {
  if ( Prewarped_Distance( x, y, x1, y1 )>1 )//如果有空洞
    Holefilling_Using_Bar( x, y, x1, y1 );//填补空洞
}
```

图 5 条纹理的选择映射加速浮雕纹理映射的伪代码

图 6 提供的实验结果图能清楚地解释条纹理选择映射生成中间图像的过程.图 6(a)是第 1 步获得的未经空洞填补的中间图像;图 6(b)是第 2 步中绘制的用于填补空洞的条纹理,它们可以用来填补图 6(a)中出现的空洞;图 6(c)是图 6(a)和 6(b)"叠加"得到的完整的中间图像.把所有条纹理颜色设为黄色,所得到的结果图 6(d)和图 6(e)可以使读者更加清楚地理解这一过程.

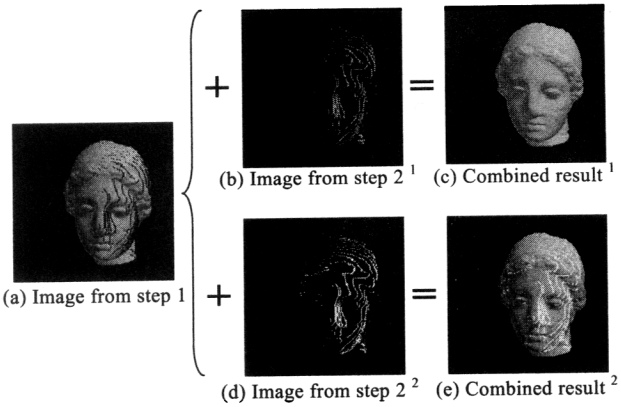

图 6 条纹理的选择映射填补中间纹理图像上的空洞的过程示意图

实验结果显示,条纹理选择映射同样能获得高质量的成像,并且把原算法的速度提高了 25%~400%不等.速度的变化取决于视点的位置,因为视点的位置将决定被选择出来填补空洞的条纹理的数目,具体讨论见第 3 节.

2.2.2 条纹理的全映射

在基于图像的绘制里,McMillan 提出的极线原则[7,8]保证了将映射到目标图像上同一位置的源图像的多个像素,能够按距离视点由远到近的顺序来处理,以正确反映场景的遮挡关系.它首先要计算出当前视点在源图像平面上的正投影,称为极点[7].在图 7 中,极点 e_r 是目标视点 a 在源图像平面上的正投影的交点.然后,源图像在极点处进行水平和垂直的分割,形成 4 个矩形区域(如图 7 中虚线所分割的区域).若极点落在源图像平面外,则源图像被分割成两个或一个矩形区域,如图 7 中视点分别在(3)区和(4)区的时候.如果目标视点位置在源视点的前方,则将按朝向极点的顺序依次处理每个矩形区域里的像素,反之则按照背向极点的方向处理,这样就能在生成的目标图像中保证正确的遮挡关系[7].

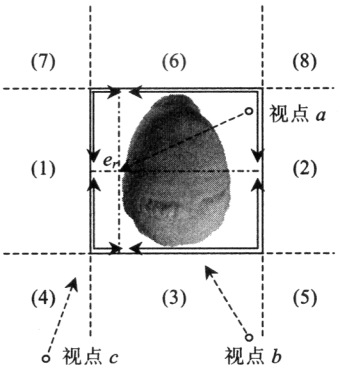

图 7 极点将源图像划分为几个不同的矩形区域

如果按照极线原则的顺序依次处理源图像所有的条纹理,即按照由远往近的顺序依次完成这些条纹理到目标成像面的映射,那么我们将能够获得该参考图像在目标成像面上完整的、连续的成像.例如,对于图 8 所示的 A,B,C,D,E 五点的条纹理,由于 E 点距离视点最远,该处的条纹理已经被 C 和 D 的条纹理所遮挡,而 C 的条纹理的上半部分也将被 B 的条纹理遮挡.如果按照由远往近的顺序依次将 E,D,C,B,A 这 5 点的条纹理映射到成像平面,那么,我们将正确地获得成像面上 A' 与 C' 之间的成像,而保证不会在 A' 与 C' 之间出现空洞.

在本文的实验里,源参考图像都是通过正投影获得的,源视点可被认为在无穷远处,所以将按照朝向极点方向的顺序依次映射源图像的条纹理,如图 9 所示.

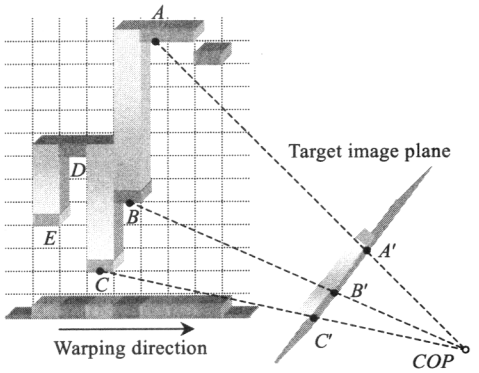

图 8 按照由远到近的顺序绘制条纹理可以保证成像正确的遮挡关系

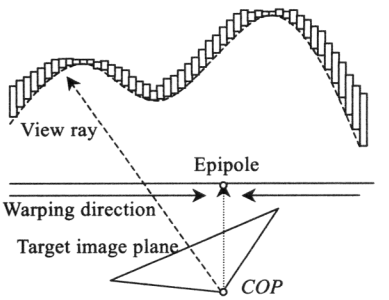

图 9 按朝向极点的顺序绘制条纹理

在基于图像的绘制里,单幅参考图像对三维场景的表达能力是有限的.对此,条纹理全映射的处理方法是,首先根据视点的位置判断当前所能看到的场景包围盒的表面.绘制时,只需对这些可见表面对应的条纹理,完成它们到成像面的条纹理映射,就可以获得完整的目标图像.图 10 显示了当视点位于模型右上方时,关于当前视点的目标图像是通过把前、上、右这 3 个可见表面对应的条纹理,按照极线原则的顺序依次映射到成像面上获得的.图 10(a)~10(c)分别对应于前、上、右这 3 幅参考图像的条纹理在成像面上的映射结果,图 10(d)是最终的条纹理全映射结果.

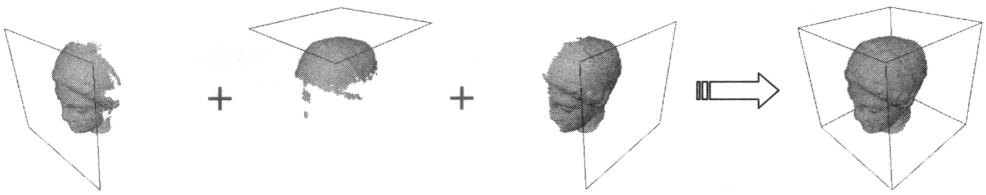

图 10 当能同时看到包围盒 3 个表面时,目标图像可以通过把 3 个可见表面的条纹理映射到成像面上获得

2.3 多层条纹理

浮雕纹理映射的一个主要缺陷是它不能正确处理具有多层几何特征的复杂场景.当视点位置发生改变使得场景中原本不可见的部分变为可见时,将会产生空洞.为此,我们结合条纹理映射和多层深度图像(layered depth images)[6]的思想,提出多层条纹理(layered depth bars,简称 LDB)的模型表示和绘制方法,即:将多个深度图像正投影到一个成像面上形成源图像时,对源图像中每个像素在各个深度图像中的对应点,根据其深度形成各自的条纹理并依次串联,得到该像素的多层条纹理(如图 11 所示).

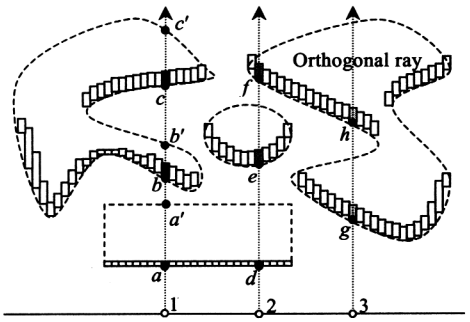

图 11 多层条纹理:对像素 1,沿其深度方向,记录场景各个层面上的条纹理 *a*,*b*,*c*;为像素 2 记录 *d*,*e*,*f*;为像素 3 记录 *g*,*h*

显然,条纹理全映射可以方便地用来对多层条纹理进行成像,只是在处理源图像的每一个点时,要将它关联的各个条纹理按照由后往前的方式依次处理即可.

3 实验结果及讨论

为了验证新方法的性能并与其他几种同类算法进行比较,我们 C 语言构建了一个统一的测试平台,同时在上面实现了条纹理的选择映射、条纹理的全映射、浮雕纹理映射[8]、硬件加速的浮雕纹理映射[9]和层次纹理映射[10]这 5 种基于图像绘制的算法.

对一个含有 33 587 个顶点和 67 170 个三角面片的维纳斯头部模型,我们为各种算法生成了分辨率大小同为 256×256 的关于该模型包围盒表面的 6 幅深度图像以及它们的层次纹理数据和条纹理数据.由于随着视点位置的变化,需要生成目标图像的参考图像的数目(即可见的包围盒表面的个数)也将发生变化,从而影响各种算法的成像效果和速度.为此,我们根据视点位置分 3 种情况讨论算法的效率.下面用 Ⅰ 区、Ⅱ 区和Ⅲ区分别表示在不同视点位置可以看到 1,2,3 个包围盒表面的情况.

我们在一台 DELL 微机上进行了实验.该计算机配有 Intel Pentium Ⅳ 2.53G CPU、512M DDR 内存、带有 128M 显存的 Nvidia GeForce Ti 4600 图形卡和装有 MS Windows 2000 操作系统.以上 5 种算法关于不同视点位置的平均帧速和部分成像结果分别如表 1 和图 12 所示.在表 2 里,我们比较了条纹理映射和层次纹理映射在预处理时间和外存储空间上的差别.

表 1 几种算法的速度比较

		Bar texture mapping		Relief texture mapping	Hardware-Assisted relief texture mapping	Layered texture mapping
		Method one	Method two			
Speed (F/S)	Region Ⅰ	98	110	18	205	55
	Region Ⅱ	38	33	20	115	43
	Region Ⅲ	23	37	18	80	35

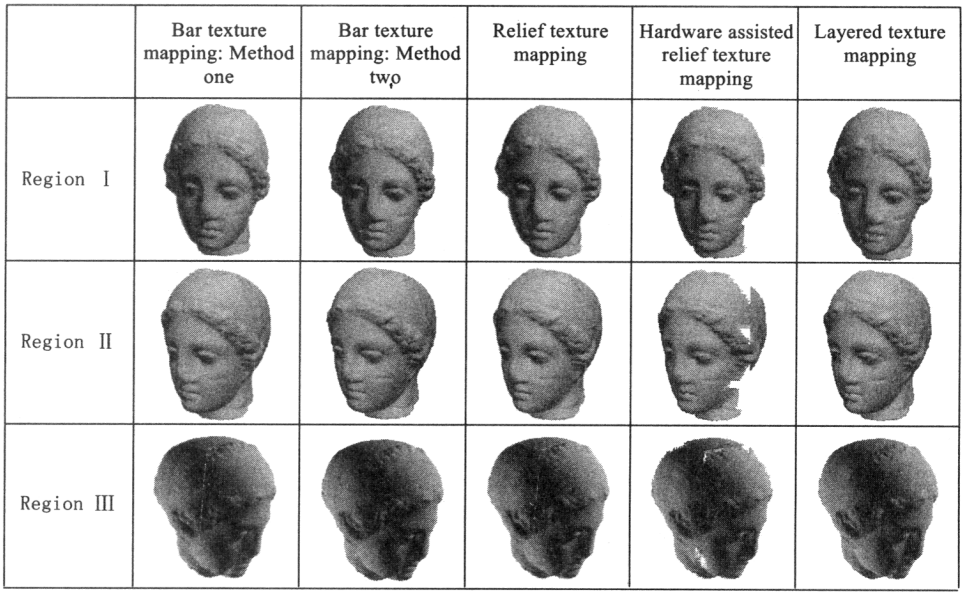

图 12 几种算法的成像

表 2 条纹理映射与层次纹理映射在预处理时间和外存储空间上的比较

	Bar texture mapping	Layered texture mapping
Pre-Processing (s)	2.641	1.953
Storage (Mb)	5.25	12.0

由实验结果可以看出,本文提出的条纹理映射可以生成高质量的成像,而且能满足实时显示的需要.在速度上,新方法要比浮雕纹理映射[8]快,且不亚于层次纹理映射[10],虽然在大多数情况下新方法要慢于硬件加速的浮雕纹理映射[9],但它的成像质量明显高于后者.在外存储空间的要求上,新方法要比层次纹理映射低很多,因而要比后者能够更广泛地应用于大规模复杂场景的显示.

对于本文提出的条纹理选择映射,被选择出来用于填补空洞的条纹理数目将影响其绘制速度.当数目很少的时候,速度可以达到 100F/S 以上,但随着视点的变化,使得用来填补空洞的条纹理数目增多时,速度会降低(如图 13 所示).这一点也可以从表 1 中的数据可以看出,当视点在Ⅱ区和Ⅲ区时,帧速要比视点在Ⅰ区时低.但新方法总是要比浮雕纹理映射快很多,无论视点在什么位置.

由条纹理映射的绘制过程可知,新方法的效率不仅与图形硬件的性能有关,还与源图像上所有扩展过的像素数目相关.被扩展的像素数目越少,即长度大于 1 的条纹理的数目越少,算法效率将越高.对于像墙壁这样比较平坦的模型,将不需要对其深度图像上过多的像素进行扩展.此时,模型的条纹理表示将基本等价于一般的二维纹理表示.对这样的模型进行条纹理映射时,将获得更快的成像速度.如图 14 所示为用条纹理全映射绘制的不同视点位置看到的几种模型的成像.对于表面平坦的骷髅背向面,条纹理全映射的绘制速度明显要比其他 5 个成像的速度快很多.

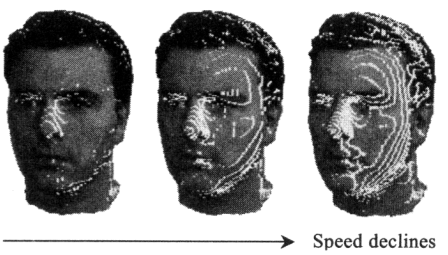

图 13 当用来填补空洞的条纹理数目增多时,会影响条纹理选择映射的效率

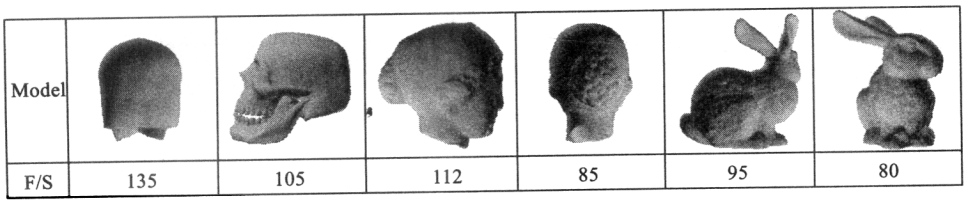

图 14 条纹理全映射绘制的几个模型的成像

图15给出了用多层条纹理和浮雕纹理映射绘制复杂场景的比较.很明显,多层条纹理技术很好地处理了可见性变化.

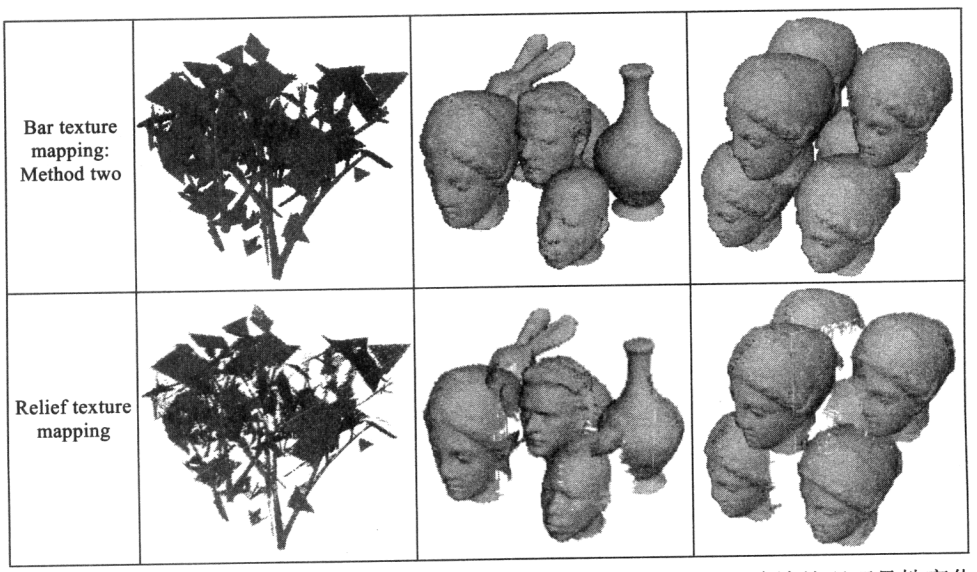

图 15 与浮雕纹理映射相比,多层条纹理在绘制层次复杂场景时能正确地处理可见性变化

4 结束语

本文提出了一种新的基于图像绘制的算法,能高效地反映模型三维凸凹细节和视差变化.它将源深度图像组织成条纹理的表达形式,构成了场景的一种近似连续的三维表示.这样,条纹理映射就可以在成像时自动完成空洞填补的工作,并且能方便利用图形硬件进行加速.与已有的同类方法相比,新方法成像速度快、成像质量高、所需存储空间小,且能方便地处理有多层几何特征的复杂场景,特别是它在处理那些比较平坦的三维模型时将更加高效.

References:

[1] Blinn JF, Wallace JR, Simulation of wrinkled surfaces. In: SIGGRAPH 1978 Proc. New York: ACM Press, 1978. 286~292.

[2] Logie JR, Patterson JW. Inverse displacement mapping in the general case. Computer Graphics Forum, 1995,14(5):261~273.

[3] Wang L, Wang X, Tong X, Lin S, Hu S, Guo B, Shum H, View-Dependent displacement mapping. In: SIGGRAPH 2003 Proc. New York: ACM Press, 2003. 334~339.

[4] Kajiya JT, Kay TL, Rendering fur with three dimensional textures. In: SIGGRAPH 1989 Proc. New York: ACM Press, 1989. 271~280.

[5] Meyer A, Neyret F. Interactive volumetric textures. In: Eurographics Rendering Workshop 1998 Proc. New York: Springer-Verlag, 1998. 157~168.

[6] Shade J, Gotler S, He L, Szeliski R. Layered depth images. In: Cunningham S, Bransford W, Cohen MF, ed. SIGGRAPH 1998 Proc. New York: ACM Press, 1998. 231~242.

[7] McMillan L, An image-based approach to three-dimensional computer graphics [Ph.D. Thesis]. University of North Carolina, 1997.

[8] Oliveira MM, Bishop G, McAllister D. Relief texture mapping. In: SIGGRAPH 2000 Proc. New York: ACM Press, 2000. 359~368.

[9] Fujita M, Kanai T, Hardware-Assisted relief texture mapping. In: EUROGRAPHICS 2002 Proc. 2002. 257~262.

[10] Wang W, Zheng X, Wu E. Layered texture mapping. In: Proc. of the Int'l Conf. on Virtual Reality and its Application in Industry. 2002. 539~546.

[11] Wynn C. Using P-buffers for off-screen rendering in OpenGL. NVIDIA Corp. White Paper, 2001. http://developer.nvidia.com/view.asp?IO=PBuffers_for_OffScreen

[12] Wynn C. Using P-buffers for off-screen rendering. In: Game Developers Conf. 2001. http://www.cs.uct.ac.za/Research/CVC/Reading/VertexShaders/PixelShaders-UsingPixelBuffers.pdf

[13] Oliveira MM, Bishop G. Factoring 3-D image warping equations into a pre-warp followed by conventional texture mapping. Technical Report, TR99-002, UNC Computer Science, University of North Carolina, 1999.

[14] Oliveira MM. Relief Texture Mapping [Ph.D. Thesis]. University of North Carolina. 2000.

[15] Wynn C, OpenGL Render-to-Texture. In: Game Developers Conf. 2002. http://developer.nvidia.com/object/gdc_oglrtt.html