山东科技大学

硕士学位论文

虚拟现实中LOD技术的研究

姓名：赵伟

申请学位级别：硕士

专业：计算机软件与理论

指导教师：郑永果

20030501

摘 要

虚拟现实(VR)、计算机辅助设计(CAD)和科学计算可视化应用通常要求能够对复杂的场景结构进行交互式显示和观察，然而这些多边形场景的计算复杂度和存储空间要求已经大大超过了当前图形硬件的承受能力。为了减少最终绘制的场景复杂度，作为实时图形生成技术重要的一部分，本文主要研究用于实时图形生成的多细节层次模型(Levels of Detail,简称LOD)的动态生成与绘制技术。

本文第一章简单介绍了VR的基本概念、重要特点以及虚拟现实系统的分类和应用领域，给出了VR中实现图形实时生成的有关方法。

第二章介绍了Garland等提出的基于顶点对折叠的简化算法，指出了该算法的一个缺陷并给出了解决方法；将Garland的算法应用到了具有一定属性的面片上，分析扩展之后算法的计算复杂度，提出了决解复杂度的方法；

第三章节给出了一种新的自动递进网格算法，可以实现无二义性，快速恢复等功能；此节提出了一个新的概念：网格密度。接下来的两部分分析了网格密度在离散多分辨率模型和连续多分辨率模型中的应用。

第四章节介绍了对BSP树的一些研究和算法的实现技术。

第五章提出了论文的后继工作，包括数据压缩、视点相关以及两种多分辨率模型的混合编程等。

关键词：虚拟现实，细节层次，网格简化，网格密度，递进网格，多分辨率模型，BSP树

Abstract

Virtual reality (VR) applications, computer-aided design (CAD) applications and scientific visualizations often need user-steered interactive displays of very complex polygonal environments, for which, however, the computation and storage requirements far exceeds the capacity of modern graphics hardware. Levels of detail (LOD) representations are an important tool for real-time rendering of such environments. This thesis contributes to the dynamic generation and rendering of models at multiple level of detail (LOD) used in real-time graphics rendering.

In Chapter 1, we describe the basic concepts and primary characteristics of VR systems, then present the methods for accelerating graphics rendering in VR.

The related research works, research contents and importance associated with LOD models are also described in chapter 2. This chapter optimized the algorithm provided by Michael Garland and Paul S.heckbert based on edge contraction, and the improved algorithm can produce better approximation.

A new method provided to simplify triangulated model with appearance attributes, and optimized the method to reduce the complexity in chapter 3. The dissertation advances a new concept: Density of Meshes, and study the applications of density of meshes in discrete multi-resolution and continuous multi-resolution.

In chapter 4, a new method is presented for implementing the BSP tree.

The future work is offered in Chapter 5, it includes the compact of data, view independence and the rendering of model with both discrete multi-resolution and continuous multi-resolution.

Keywords: virtual reality; LOD; mesh simplification ;mesh density; process mesh; multi-resolution model.

声　明

本人呈交给山东科技大学的这篇硕士学术论文，除了所列参考文献和世所公认的文献之外，全部是本人在导师指导下的研究成果。该论文尚没有呈交于其它任何学术机关作鉴定。

研究生签名：

日　　期：

AFFIRMATION

I declare that this dissertation, submitted in fulfillment of the requirement for the award of Master of Philosophy, in Shanddong University of Science and Technology, is wholly my own work unless referenced of acknowledge. The document has not been submitted for qualification at any other academic institute.

Signature:

Date:

第 1 章 虚拟现实系统简介

1.1 基本概念

虚拟现实(Virtual Reality，简称 VR)，又译为临境、灵境。它是由美国 VPL Research Inc。公司的 J.Lanier 在 1989 年提出的一个词，通常是指用头盔显示器和传感手套等一系列新型交互设备构造出一种计算机软硬件环境，人们通过这些设备以自然的技能（如头的转动、身体的运动等）向计算机送入各种命令，并得到计算机对用户的视觉、听觉及触觉等多种感官的反馈。随着人们动作的变化，这些感官反馈也随着改变。但是现在与虚拟现实有关的内容已经扩大到与之相关的许多方面，像“人工现实”（Artificial Reality）、“遥在”（Telepresence）、“虚拟环境”（Virtual Environment）等。

关于虚拟现实的定义，许多人有不现的看法，一般认为虚拟现实是一种新的人－机界面，它为用户(参与者)提供了一种具有临场感和多感觉通道的体验，试图寻求一种最佳的人－机通讯方式。

VR 技术是一种高度逼真地模拟人在自然环境中视、听、动等行为的人机界面技术。这种模拟具有两种基本特征，即“沉浸”特征和“交互”特征。VR 的“沉浸”特征要求计算机所创建的三维虚拟环境能使“参与者”得到全身心置于环境之中的体验。VR 的“交互”特性主要是指参与者通过使用专用设备实现用人类自然技能对虚拟环境中的实体进行交互考察与操作。因此，VR 技术将从根本上改变人与计算机系统的交互操作方式。

1.2 VR 系统概述

1、VR 系统的特性

虚拟现实系统就是要利用各种先进的硬件技术及软件包件工具，设计出合理的硬件、软件及交互手段，使参与者能交互地观察和操纵系统生成的虚拟世界。从概念上讲，任何一个虚拟现实系统都可以用三个“I”来描述其特性，这就是“沉浸(Immersion)”、“交互(Interaction)”和“想象（Imagination)”。这三个“I”反映了虚拟现实系统的关键特性，就是系统与人的充分交互。虚拟现实系统的设计要达到以下几个目标：

首先，要使参与者有真实的体验。这种体验是“沉浸”或“投入”，即全身心地进入，简单地说就是产生在虚拟世界中的幻觉。理想情况下虚拟环境应该达到使用户难以分辨真假的程度，甚至比真的还要真实。

其次，系统能提供方便的、丰富的、主要是基于自然技能的人机交互手段。这些手段使能参与者能够对虚拟环境进行实时的操纵，能从虚拟环境中得到反馈的信息，也能使系统了参与者关键部位的位置、状态。变形等各种需要系统知道的数据。

最后，因为虚拟现实不仅仅是一种媒体或用户的高端接口，而且还是针对某一特定领域。解决某些问题的应用，为了解决这些问题，不仅需要了解应用的需求，了解技术的能力，而且还需要有丰富的想象力。

交互和沉浸感是任何虚拟现实经历的两个实质性的特性。

2、VR 系统的发展史

VR 技术并非最近几年才出现，它的起源要追溯到计算机图形学之父 Ivan Sutherland 于 1965 年在 IFIP 会议所作的标题为“The Ultimate Display”的报告。在该报告中，Ivan Sutherland 提出了一项富有挑战性的计算机图形学研究课题。他指出，人们可以把显示屏当作一个窗口来观察一个虚拟世界。其挑战性在于窗口中的图像必须看起来真实，听起来真实，而且其中物体的行为也很真实。这一思想奠定了 VR 研究的基础。1968 年，Ivan Sutherland 发表了题为“A Head--Mounted 3D Display”的论文，对头盔式三维显示装置的设计要求、构造原理进行了深入的讨论。Sutherland 还给出了这种头盔式显示装置的设计原型，成为三维立体显示技术的奠基性成果。

VR 研究的进展从六十年代到八十年代中期是十分缓慢的。直到八十年代后期，VR 技术才得以加速发展。这是因为显示技术已能满足视觉耦合系统的性能要求，液晶显示(LCD)技术的发展使得生产廉价的头盔式显示器成为可能。

VR 系统的最大特点就是参与者能与计算机生成的虚拟环境进行自然的交互，能用人类自然的技能与感知能力与虚拟世界中的对象进行交互作用。设想这样一个虚拟现实系统的情景：用户在戴上头盔显示器和数据手套后，不仅可以看到一个排球，而且还可以摸到它，并能得到触觉反馈。用户还可以拍动排球，拍球的时候，不仅能感受到排球对手的反作用力，还能听到拍球时发出的“嘭嘭”声，这将会令人产生多么真实的感受。与传统的人－机界面系统相比，虚拟现实的沉浸式交互技术是一种全新的人机交互风范。在传统的人－机系统中，用

户是一个外部的观察者，只是通过显示屏这个范围很有限的小窗口，观察计算机内的合成环境，而 VR 系统则是要用户参与到这个合成环境中。传统的进行人－机交互最常用的设备是键盘和二维鼠标，人们通过它们与合成环境中的物体进行通讯，这与我们在自然世界中的通讯方式相距甚远。

为了使用户产生身临其境的感觉，VR 系统必须具备一些不同于传统人－机系统的交互技术。这些能使人身临其境的交互技术主要有：大视角的立体显示、头部跟踪、人及姿势跟踪、三维声音、触觉反馈和力反馈等技术。目前，实现这些交互技术的设备主要有：头盔显示器、数据手套、三维位置传感器和三维声音产生器等。

VR 系统可分为三大类：桌面 VR 系统、沉浸式 VR 系统和分布式 VR 系统。

桌面 VR 由于采用标准的 CRT 显示器和立体显示技术，其分辨率较高，价格较便宜。在使用时，桌面 VR 系统设定一个虚拟观察者的位置。桌面 VR 系统通常用于工程 CAD、建筑设计以及某些医疗应用。沉浸式 VR 系统利用头盔显示器把用户的视觉、听觉和其它感觉封闭起来，产生一种身在虚拟环境中的错觉。分布式 VR 系统则是在沉浸式 VR 系统的基础上将不同的用户通过网络联结在一起，共享同一个虚拟空间，使用户达到一个更高的境界。

很早以前，人们已开始设想用一种新设备来产生能替代真实环境体验的新机器，但早期的 VR 系统与计算机并无多大关系，只有到计算机技术比较成熟的今天，才使虚拟现实成为现实。

3、VR 系统的应用

由于虚拟现实在技术上的进步与逐步成熟，其应用在近几年发展迅速，应用领域已由过去的娱乐与模拟训练发展到包括航空、航天、铁道、建筑、土木、科学计算可视化、医疗、军事、通讯等广泛领域工。下面介绍 VR 已有的一些应用成果。

(1) 医疗和康复

近年来，计算机在医学界的广泛应用(如病人数据库、手术模拟、远程咨询、数字化 X 射线照片、专家系统等)，改变了人们医疗和康复的模式，特别是 VR 技术在医疗和康复中的应用，将根本改变这一方式。目前，VR 技术在这一领域的具体应用系统及有关研究包括：

- 虚拟解剖训练系统，该系统主要用于教学。
- 外科手术模拟系统。
- 遥控手术，其原型系统 SRI 远程手术模拟系统也已开发成功。
- 手功能诊断，利用数据手套来检查病人手功能失调的程度。如 Greenleaf 医疗系统公司

的运动分析系统，就是一个商品化的例子。

- 虚拟训练机，是帮助病人康复训练的 VR 系统。
- 帮助残疾人改变生活方式的有：轮椅虚拟现实系统、会说话的手套、聋哑人电话等。
- 用于新药试制的三维分子结构显示与操纵的 VR 系统，有纽约大学与 Division 公司联合开发的 Grope II 等。

(2) 娱乐、艺术和教学

在 VR 技术发展的早期，娱乐是 VR 技术发展的主要和直接推动力，而现在又成了 VR 系统的主要市场之一。在公共场所获得成功的 VR 游戏项目有：世界上第一个大型 VR 娱乐系统 Battle Tech 中心，娱乐中心于 1990 年在芝加哥开张，可供数十人共同玩战争游戏。此外，还有 Hughes 飞机公司和 LucasArts 娱乐公司等开发的 Mirage VR 模拟器。W。Industries 有限公司于 1991 年开发了世界上第一个采用头盔显示器的娱乐系统，称为 Viruatily System，并获 1992 年 VR 产品奖。

VR 技术在未来的艺术领域将扮演重要的角色。它作为一种新的媒体不仅能变静态艺术为动态艺术，而且将成为联结艺术创作者与欣赏者的重要纽带。目前，与艺术有关的应用系统有：Videoplace、Mandale、虚拟演员(Virtual Actors)、虚拟博物馆(The Virtual Museum)、虚拟音乐(Virtual Music)等。

VR 技术在教育领域特别是在中小学教育中的作用，目前尚处于研究证实阶段，但其前景是十分诱人的。由 Loffin 及其同事开发的虚拟物理实验室，是一个用于演示牛顿力学及量子物理有关定律的虚拟实验教学系统。

(3) 军事和航空航天

在军事和航空航天领域，模拟和训练特别重要。随着当今技术的高度复杂化，硬件系统的开发周期不断缩短，迫切需要一种灵活的、可升级的、网络化的模拟系统。VR 技术在上述各个方面均能很好地满足这一需要。因此，美国政府充分认识到 VR 技术在保持美国技术领先方面具有的战略意义，并制定了一系列的实施计划。其中应用于军事方面的 VR 系统有：

- 坦克训练网络 SIMNET。
- 虚拟毒刺导弹训练系统。
- 反潜艇作战系统 ASW。

在航空航天领域的 VR 系统有：

- NASA 虚拟现实训练系统。

- EVA 训练系统。
- 虚拟座舱。

(4) 商业应用

VR 技术在商业领域最早也是最成功的应用是产品广告宣传。VR 广告比传统广告更易于制作和更改，也更具有感染力。由此，许多公司已认识到率先使用 VR 技术可使他们的产品领导潮流。已有的一些应用包括：

- Maxus 证券交易可视化系统，如哥伦比亚大学研制的 n-Vision 系统。
- 体验广告，这类广告在美国多附着于公共场所的 VR 娱乐设施；在欧洲则开发出不少独立的 VR 广告系统，如 Kingston Micro Electronies、Callscan 、Madrid City Council 等。
- 室内装潢设计，如 Light-space Software 公司开发的灯饰可视化系统 LVS(Lightcape Visualization System)和 LEL 模拟系统(The Living Environmental System)等。

(5) 自动控制和制造业

VR 技术一旦全面应用于自动控制和制造业，必将产生巨大的经济效益。但是由于该领域固有的技术复杂性，VR 技术在这一领域的应用仍处于早期的研究和开发阶段，比如：

- 机器人辅助设计。
- 脱机编程，利用 VR 技术仿真机器人，编程调试不影响控制过程，大大减少了时间损耗。
- 远程操作，可提供对远程恶劣环境控制系统的精确控制。

在制造业领域，VR 技术可能将零星替代并超过目前广泛使用的 CAD 系统。它可将产品需求分析、时间成本分析和产品设计，甚至将相应的生产线设计集成在一起，以进一步缩短新产品的研制开发周期，降低成本。特别是对于产品定制和售后服务，VR 技术的应用具有更大的优势。

总之，随着 VR 技术的进一步成熟，其应用领域和应用深度将不断发展。

1.3 课题的提出及本文所做的工作

本文研究的是虚拟现实中 LOD(Level of Details)简化算法，所谓 LOD 模型方法，即为每个物体建立多个相似的模型，不同模型对物体的细节描述不同，对物体细节的描述越精确，

模型也越复杂。根据物体在屏幕上所占区域大小及用户视点等因素，为各物体选择不同的LOD模型，从而减少需要显示的多边形数目。在选择删减多边形时，国内外的学者提出了许多简化方法，就大体来讲可以分为三类：基于多边形删减、基于边的删减和基于点的删减。本文所讨论的算法是基于边的删减（又称为边折叠）算法。

1、问题的提出

论文所讨论的基本算法是由 Garland 等人提出的基于二次方误差的边折叠算法，该算法能够产生很好的简化结果，具有高效通用的特点，是目前广泛使用的简化算法，但此算法中尚有一点不足，在一定的条件上，简化算法会错误地选择简化边，导致对原模型的简化不够精确，并且原算法不支持对具有不同属性的三角面片的简化。

2、本文所做的工作

本文在实验的基础上找到此算法的不足并加入改进，改进后的算法能够产生比原算法更为精确的近似模型。本文将原算法推广到对具有各种属性的三角面片的简化，并且分析了推广后的算法所表现出的高计算复杂度问题，根据矩阵理论找到一种能够减少计算复杂度的方法。

递进网格是 LOD 模型在实际应用的一种有效推广，递进网格给出了一种能够自动产生LOD层次模型的方法。本文的第三章提出了一种新的概念：网格密度，通过网格密度可以产生符合客观规律的LOD层次，并能很好的支持递进网格。接下来讨论了网格密度在两种不同的多分辨率模型中的应用情况：在离散的多分辨率模型中通过用网格密度可以观察到任意层次的简化模型，我们称之为准连续多分辨率模型；在连续多分辨率模型中，网格密度可以很好地应用在地形的简化过程中。

第四章对BSP树作了初步的研究，给出BSP树的定义、特点及在LOD中的应用，最后给出了一种BSP树的创建和绘制方法，并给出了实验结果。

第2章 多细节层次模型

2.1 基本概念

2.1.1 LOD的提出

就目前计算机图形学水平而言，只要有足够的计算时间，就能生成准确的像照片一样的计算机图像。但VR系统要求的是实时图形生成，由于时间的限制，使我们不得不降低虚拟环境的几何复杂度和图像质量，或采用其它技术(如纹理映射)来提高虚拟环境的逼真程度。

所谓实时显示，是指当用户的视点变化时，图形显示速度必须跟上视点的改变速度，否则就会产生迟滞现象，要消除迟滞现象，计算机每秒钟必须生成10帧到20帧图像，当场景很简单时，例如仅有几百个多边形，要实现实时显示并不困难，但是，为了得到逼真的显示效果，场景中往往有上万个多边形，有时多达几百万个多边形。此外，系统往往还要对场景进行光照处理、反混淆处理及纹理处理等等，这就对实时显示提出了很高的要求。

就图形学发展而言，起关键作用的无疑是图形硬件加速器的发展。高性能的图形工作站和高度并行的图形处理硬件与软件体系结构是实现图形实时生成的一个重要途径。然而应用模型的复杂程度往往超过当前图形工作站的实时处理能力，考虑到VR对场景复杂度几乎无限制的要求，在VR高质量图形的实时生成要求下，如何从软件着手，减少图形画面的复杂度，已成为VR中图形生成的主要目标。

要提高图形显示速度，一个实践证明非常有效的方法是降低场景的复杂度，即降低图形系统需处理的多边形数目。目前，比较常用的方法有下面几种：

预测计算：该方法根据各种运动的速率和加速度，如人体头部转动速度，用预测、外推法在下一帧画面绘制之前估算出头部跟踪系统及其它输入设备的输入，从而减少由输入设备所带来的延迟。

脱机计算：由于VR系统是一个多任务的模拟系统，所以有必要尽可能地将一些可预先计算好的结果预先计算并存储在相应的结构中，其中包括全局光照模型、动态模型的计算等。

场景分块：一个复杂的场景可被划分成多个子场景，各子场景之间几乎不可见或完全不可见。例如把一个建筑物按房间划分成多个子部分。此时，观察者在某个房间仅能看到房内

的场景及与门口、窗户等相连的其它房间，这样，系统就能有效地减少在任意时刻需要显示的多边形数目，从而有效地降低了场景复杂度。但是，这种方法对封闭空间有效，对开放空间则很难使用这种方法。

可见消隐：与场景分块方法不同，这种方法与用户的视点关系密切，而前者仅与用户所处场景位置有关。使用这种方法，系统仅显示用户当前能“看见”的场景，当用户仅能看到场景的很少一部分时，由于系统仅显示相应场景，从而大大减少了需要显示的多边形的数目。然而，当用户能“看见”的场景较复杂时，这种方法也不起作用。

细节层次模型：即使采用了场景分块技术及可见消隐技术，有时用户能“看见”的场景仍很复杂，为此细节层次(Level of Details， 简称 LOD)模型方法应运而生。所谓 LOD 模型方法，即为每个物体建立多个相似的模型，不同模型对物体的细节描述不同，对物体细节的描述越精确，模型也越复杂。根据物体在屏幕上所占区域大小及用户视点等因素，为各物体选择不同的 LOD 模型，从而减少需要显示的多边形数目。这是一种有前途的方法，然而，这种方法对场景模型的描述及维护提出了较高的要求。

比较以上几种方法，前面几种方法仅适用于某些特殊的情况，而 LOD 模型方法则具有普适性。现在，LOD 模型的自动生成和绘制技术已成为一个很有前途的研究方向，受到了全世界范围内相关研究人员的重视。

2.1.2 LOD 的发展

早在计算机发展的初期，人们已经认识到层次模型表示的重要性。Clark 在 1976 年就已

图 2.1.1：三角形面片数目均匀递增的一组多细节层次模

提出了用于可见面判定算法的几何层次模型，这是因为三维物体和环境定义中固有的几何结构性质，不仅可用来定义物体的相对运动和所处位置，而且有助于解决图形学中许多其它相

关的一些问题[7]。实际场景的复杂性往往超过计算机能进行实时处理的能力，为了在计算机中以较少的时间及较高的图像质量绘制出复杂的场景，Rubin 提出了复杂场景的层次表示方法及相关的绘制算法。Marshall 等人提出的生成三维地形的过程建模方法，也包含了细节层次的思想。然而，前面所述的层次模型一般都是人工建立的。

九十年代初，图形学方向上派生出虚拟现实和科学计算可视化等新研究领域。虚拟现实和交互式可视化等这一类交互式图形应用系统要求图形生成速度达到实时，而计算机所提供的计算能力往往不能满足复杂三维场景的实时绘制目的，因而研究人员提出多种图形生成加速方法[6]，LOD 模型则是其中一种主要方法。这几年在全世界范围内形成了对多 LOD 自动生成技术的研究热潮，并且取得了很多有意义的研究结果。

在多细节层次模型自动生成方面，国际上已开展了一些研究工作。Schroeder 等人提出了基于顶点移去的模型简化方法[27]。该方法首先利用各顶点的局部几何和拓扑信息将各顶点分类，然后根据不同顶点的评判标准决定该顶点是否可以删除。如果可以删除，则采用递归环分割法对删除顶点后所留下的空洞进行三角剖分，否则保留该顶点。Hamann 给出了一种基于三角形移去的模型简化方法。该方法首先对给定模型中的所有顶点计算其曲率，然后由顶点的曲率决定所有三角形的权值，权值最小的三角形被移去，受移去三角形影响的区域则被重新局部三角剖分，对所有新引入的三角形计算其权值，重复以上过程，直到移去了所要移去三角形的数目为止。

Turk 给出了基于网格重新划分的多边形模型简化方法。该方法首先把一组新顶点分布到原模型上，然后新老顶点形成中间网格，通过必要的拓扑结构一致性检查移去老顶点，并进行局部的三角剖分操作，形成新的与原模型具有相同拓扑结构的网格模型。为了给曲率变化较大的区域分布更多的新顶点，以使重新划分的三角形网格模型较精确地反映出原模型，该算法还根据各顶点的曲率近似计算，用三角形顶点曲率与三角形面积的乘积作为权因子决定新顶点的放置，并根据顶点间的排斥半径将新顶点定位在原多边形内。

Kalvin 给出了面片合并方法自动生成物体的简化模型[28]。该方法首先以原模型中的任一多边形作为“种子”面片，根据合并条件不断合并种子面片周围的面片，直到周围面片不再满足合并条件。同时，算法根据用户定义的误差给出种子面片的近似面片集合，并根据不断合并的种子面片修改近似面片集合，直到集合为空。

以上方法都要求生成的简化模型保持拓扑关系和几何关系的正确性。但 Rossignac 认为，

简化模型只要其图形效果与原模型一致，则该简化模型成立。为了加快实时图形生成，Rossignac 还提出了一种多分辨率近似法自动生成物体的简化模型。该方法首先赋予各顶点一个权值，给物体特征变化较大处的顶点以较大的权值。然后根据模型的复杂程度、相对大小等将此物体所占空间划分为立方体的单元。同一单元中的顶点，以各顶点的权值计算这一单元中所有顶点的代表点，然后依据原模型中各多边形顶点的代表点是否为同一个代表点合并各多边形。

另外，Hoppe 采用了能量函数最优化的网格简化方法[10]并提出了累进网格的生成方法[12]。Ecke 等人则利用小波变换技术将多面体模型表示为多分辨率形式，由此可以自动生成一系列连续的原模型的近似模型[29]。该方法产生的近似模型与原模型的误差可控。

最近，国外已有一些学者提出了误差可控的三角形网格模型的简化方法，比如：Bajaj 等人提出了通过误差传递方法来度量简化模型与原模型的误差程度，从而得到误差可控的简化模型。Klein 等人提出了基于 Hausdorff 距离来度量简化模型与原模型的误差，实现了有效的误差可控的网格简化新方法[30]。

在多细节层次绘制技术研究方面，美国加州大学伯克利分校的 Funkhouse 等人把不同细节层次的场景和不同精度的绘制算法结合起来进行研究，提出了一种自适应显示算法，用该算法生成的图像能满足稳定的帧速率，在虚拟环境中获得成功应用[31]。美国佐治亚技术学院的 Lindstrom 针对规则网格表示的地形模型提出了一种实时连续细节层次绘制方法，在三维地形仿真显示中得到了应用[32]。

以上讨论的绝大多数网格模型的多细节层次自动生成技术和绘制技术都基于静态的 LOD 模型，在实际应用中，面对场景中模型大小、种类、网格结构以及观察位置等诸多细节，仅仅利用原有的多细节层次绘制方法是不够的。Xia 针对科学计算可视化观察的特殊性对大模型进行局部的细节层次调整[33]，在 1997 年，Hoppe 对累进网格[12]进行扩展，形成对任何网格都适用的且与视点有关的动态 LOD 模型。Luebke 也同时提供了另一种基于层次结构的 LOD 模型[34]，这些动态的多细节层次模型适用各种网格简化算法，在场景绘制中可以根据观察者位置的转变自动调节模型各个部位的网格细节层次，达到大量删除模型面片的目的。

国内在以上两方面也开展了一些研究工作。例如，对已有的网格简化算法进行改进或提出新的多面体模型简化方法[36][38]，给出了虚拟环境中恒帧速率自适应显示算法等等。

2.2 Garland 简化算法

2.2.1 算法简介

1、算法简介

1977 年 Michael Garland 在前人的基础上提出了一种基于边折叠的顶点对简化算法，此算法提出之后很快便成了模型简化的主要方法。此算法的优点是通过简化产生的层次结构可以有效的生成多分辨率表面模型，并能用于实时视相关 LOD 模型简化。算法利用二次方误差估计来度量简化模型的近似程度，并用一种更有效的平面集隐含表示近似误差。该算法能快速有效地产生高质量的近似结果，因此是目前最快的模型简化算法。下面对算法做一简单的介绍。

这种方法是基于边的合并（或边的折叠）的简化方法，此算法根据权值将边分类，通常权值由边折叠后产生的某种误差来决定，简化过程中，将权值最小的边简化成一个点，同时更新与生成点相关的数据。边折叠算法可以产生较好的结果，并且可以生成多分辨率模型。边的折叠过程如图 2.1 所示。

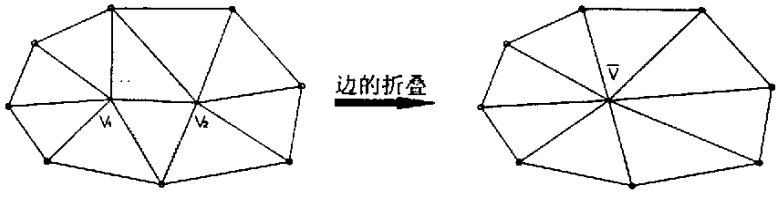


图2.1：边的折叠操作

算法由 Michael Garland 和 Paul S.Heckbert 提出[6]。算法基于边的折叠算法，二次方误差的定义在高速度低质量和高质量低速度之间做出了很好的折衷。边(v_1, v_2)的折叠可以由三步完成：

(1) 将点v_1和v_2移到新点$\bar{v}$；

(2) 将与v_2相关的边和面加到v_1中；

(3) 删除v_2点及相关信息。

图 2.1 解释了一条边的折叠操作，这个过程记为$(v_1, v_2) \rightarrow \bar{v}$。

2、 顶点对的选择

既然算法是基于边的折叠，边的选择决定了算法是否是拓扑可变的，在 Michael Garland 和 Paul S.Heckbert 的论文中将其归为拓扑可变的，于是简化算法不仅仅对原始模型中的边进行操作，对相距较近的两点也进行操作，顶点对的选择如下：

(1) (V_1,V_2) 是一条边；

(2) (V_1,V_2) 不是一条边，但 $\|V_1-V_2\|\le t$。

其中，(V_1,V_2) 代表顶点对，$\|V_1-V_2\|$ 表示两点间的距离，t 是一个阈值，它的取值将会影响简化的程度。从顶点对的选择来看，该方法是一种拓扑可变的简化方式。如果 t 取 0，则算法就变为标准的边折叠算法，并且是拓扑不变的。

3、 二次方误差的定义

不同的边折叠算法之间的差异在于误差的计算，算法将每一个顶点都赋予一个误差值，顶点 v 的二次方误差定义如下：

$$Error(v)=v^TQv \tag{2.2.1}$$

其中，$v=(x,y,z,1)$，表示顶点的齐次坐标，Q 是一个 (4×4) 矩阵。每一个点在初始化时都有一个矩阵与之对应，以产生误差值。下面说明 Q 的计算过程。

考察原始的三维模型，每一个顶点都是一组三角形面片的交点，于是我们将每一个顶点 v 和特定的三角形集合 $T=\{f_1,f_2,\cdots f_k\}$ 关联起来。我们知道，空间中任一平面可以定成下面的形式：

$$n^Tv+d=0 \tag{2.2.2}$$

式中 n 是平面的法矢量（为计算方便，我们将法矢量写成单位矢量），d 是任一常数。空间中任一点 $v(x,y,z)$ 到此平面的距离的平方可以写成：

$$D^2=(n^Tv+d)^2=(n^Tv+d)(n^Tv+d)=v^T(nn^T)v+2dn^Tv+d^2 \tag{2.2.3}$$

用一个矩阵 Q 来表示 D^2，则 Q 可写成：

$$Q=(A,b,c)=(nn^T,dn,d^2) \tag{2.2.4}$$

如果用奇次坐标来说明，则会得到另外一种更易理解的形式，如下：

把顶点坐标写为：$v=(x,y,z,1)$，则

$$\begin{aligned}\Delta(v)=\Delta(x,y,z,1)&=\sum_{n\in planes(v)}(n^Tv)^2=\sum_{n\in planes(v)}(n^Tv)(n^Tv)=\sum_{n\in planes(v)}v^T(nn^T)v\\&=v^T(\sum_{n\in planes(v)}nn^T)v=v^T(\sum_{n\in planes(v)}K_p)v\end{aligned} \tag{2.2.5}$$

在这里

$$K_p=nn^T=\begin{bmatrix}a^2 & ab & ac & ad\\ ab & b^2 & bc & db\\ ac & bc & c^2 & cd\\ ad & bd & cd & d^2\end{bmatrix} \tag{2.2.6}$$

于是Q便是所有属于集合T的面的K_p之和，用这种方式表示的Q是一个(4×4)的矩阵。

4、 算法流程

计算好每一顶点的误差值之后，我们在简化过程中对误差满足一定条件的顶点对作合并工作，算法的流程如下：

(1) 首先计算每个顶点的Q值；

(2) 选择有效的顶点对；

(3) 计算有效顶点对(v_1,v_2)合并之后的顶点$\bar{v}$，并且有$Q(\bar{v})=Q(v_1)+Q(v_2)$；

(4) 将误差满足条件的顶点合并；

(5) 如简化后的模型满足要求则结束，否则转2。

算法具有以下几个特点：

高效：算法能快速地简化复杂模型。举例来说，将一个有7000个三角面片的模型简化成有100个模型只需15秒（数据取自文献[15]），此外，模型的每个顶点只需10个浮点参数。

高质量：算法的简化结果对原始模型保持了很高的精确度，简化之后原模型的基本特征都

被保留了下来。

通用性：该算法有仅可以对边进行简化，对距离较近的顶点对同样可以简化。

算法数据如下：(取自文献[15])

表 2.2.1：不同模型的算法时间

模型名称	三角形数目	初始化时间 (ms)	简化时间(ms)
Puppy	69451	3.3	12.0
Cow	5804	0.22	0.69
Foot	4204	0.16	0.41

图 2.1.1 是算法的简化效果。

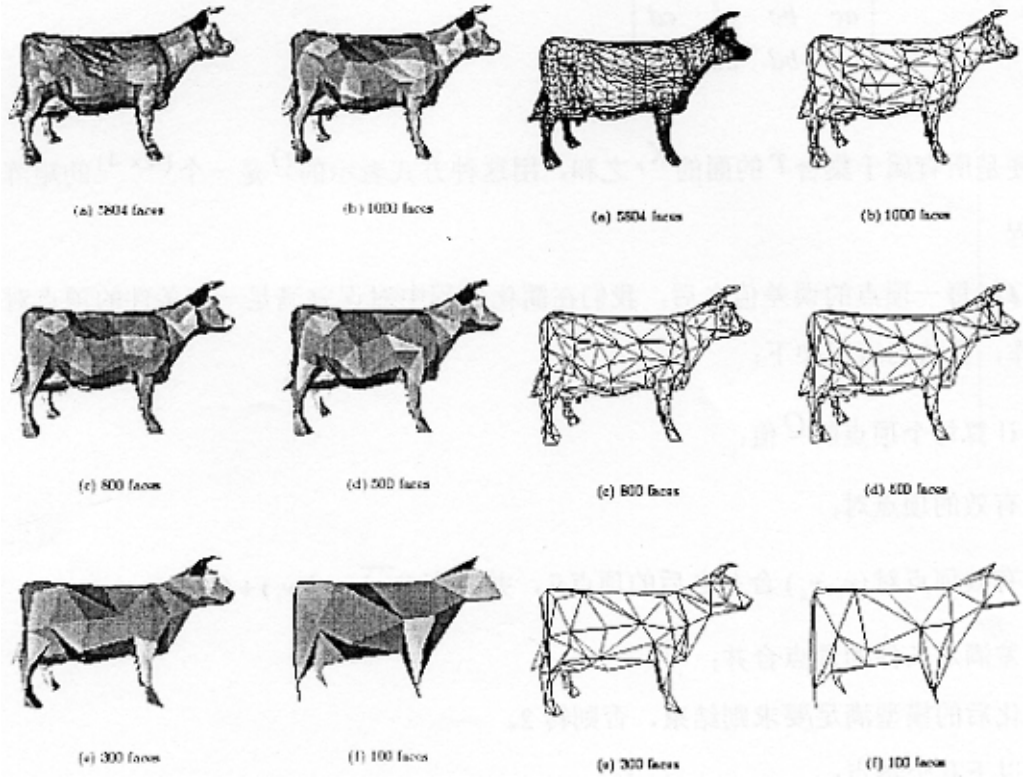


图 2.1.1 Garland 简化算法效果图

2.2.2 算法的不足及改进

1、问题的提出

Garland 简化算法高效、精确、通用性强，是目前广泛使用的简化算法，但此算法中尚有一点不足，在一定的条件上，简化算法会错误地选择简化边，对原模型的简化不够精确。

在顶点对的简化前后，与简化边相关的三角形数目有了明显地变化，见图 2.2.1 所示：

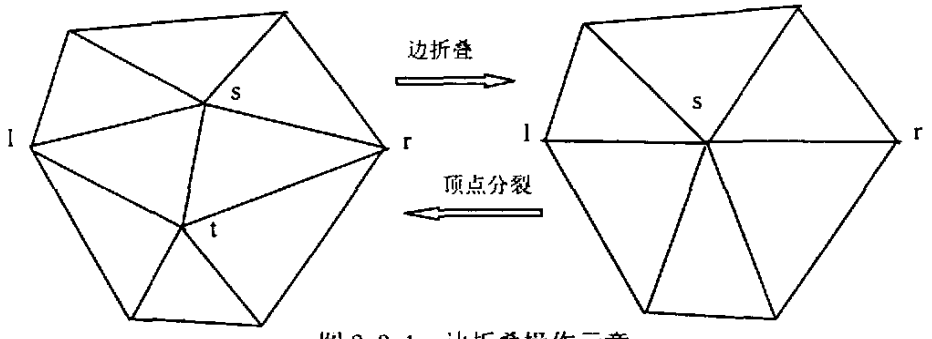


图 2.2.1：边折叠操作示意

一般情况下，假设 V_1 点和 V_2 点都不是边界点（边界上的点不给予简化，以便保持原始模型的边界特征）。用 $(V_1,V_2)\rightarrow V$ 表示将边 (V_1,V_2) 简化成新的顶点 V，在这一过程中，与顶点相关联的边的数目有了变化，假设用 $FaceNum(V_i)$ 来表示与 V_i 点相邻的三角形的数目，则变化遵循下面的公式：$FaceNum(V)=FaceNum(V_1)+FaceNum(V_2)-4$。在三维模型中，以某个点为顶点的三角形少则五六个，多则十几个，这样，在简化过程中，一个点周围的三角形为越来越多。按上一节给出的基本算法，顶点 V 的二次方误差是 V 到与 V 相关联的各个三角形的距离之和。由于 Garland 算法生成的新的三角形与简化前的三角形位置相差很小（精确性），新三角形与简化前的三角形的误差也不大，这样新生成的顶点的误差会由于与之相关联的三角形数目的增大面增大。随着简化层次的不断加深，这种趋势越来越明显。我们看一下下面图形的简化过程。

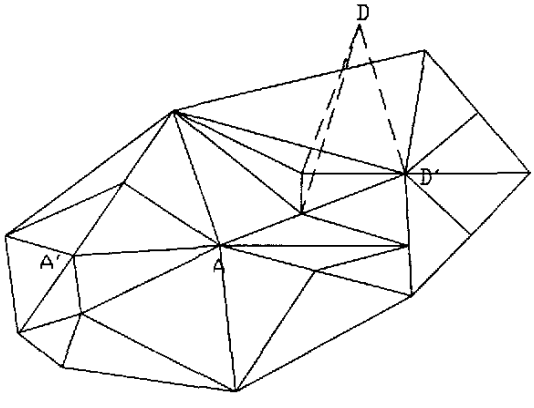


图 2.2.2 待简化的三角模型

在上图表示的模型中，有一个表示物体特征的突出（虚线表示）。就物体模型而言，越是变化大的地方或是越突兀地方越能体现物体的表面特征，这些地方，简化算法应尽量保持原形，最好不要作处理，以免影响精确程度。我们看一下 Garland 算法对边 (A,A') 与边 (D,D') 的处理过程：

在算法的一开始，A 和 A' 的母结点的误差比 D 和 D' 母结点(在折叠之前的顶点)的误差小，因为就一般情况而言，在建立原始模型时，我们并不提倡把它做得十分的复杂，以便在修改时保持一定的灵活性，这样 A 和 A' 的母结点的相关三角形数目并不多，随着简化层次的加深，A 和 A' 的母结点的相关三角形的数目会大大增加，而 D 和 D' 相关三角形数目因为 D 和 D' 没有得到简化而保持不变，它们的二次方误差也不会发生变化。假设到某一层次，形成上图所示的情况，这时 A 和 A' 的二次方误差已经足够大到比 D 和 D' 的二次方误差还大，这样在下一次简化时，便开始简化边 (D,D')，而不是边 (A,A')，但这正是我们不想看到的。所以有必要对原来的算法加以改进，以适应上面描述的情况。

2、 问题的解决

Garland 算法求得的二次方误差是用和来表示的，这是产生上述问题的基本，如果将和改为均值，问题就不会出现了。我们还是用上面的例子说明：

在形成图 2.2.2 所示的情形时，原算法中计算的边 (A,A') 的误差大大增加，但其相关的三角形数目也大大增加，如果将原来算法中边 (A,A') 的误差除以其相关三角形的数目，结果还是比较小的，这样边 (A,A') 的误差就不会大于边 (D,D') 的误差，简化程序会继续简化边 (A,A') 而不是简化边 (D,D')。改进后的流程如下：

(1) 首先计算每个顶点的 Q 值；

(2) 选择有效的顶点对；

(3) 计算有效顶点对 (V_1,V_2) 合并之后的顶点 $\overline{V}$，并且有

$$Q(\overline{V}) = (Q(V_1) + Q(V_2)) / FaceNum(\overline{V})$$；

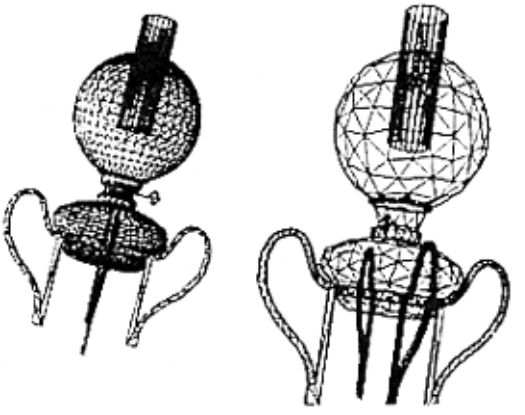
(4) 将误差满足条件的顶点合并；

(5) 如简化后的模型满足要求则结束，否则转 2。

误差计算过程的改变不会引起简化质量的改变，算法改进后可以产生和原算法同样优质的简化效果，只是对边的选择更为精确。当原始模型表面比较光滑，起伏较小时，改进后的

算法和原算法产生一模一样的近似模型[见图 2.2.3]。当原始模型表面起伏较大时，改进的算法会产生更加合理的近似模型[见图 2.2.4]。

图 2.2.3：改进后的算法同样可以产生高质量的简化模型



(a) (b)

图 2.2.4： 改进后的算法会产生更加合理的近似模型

图 2.2.3 和图 2.2.4 使用的是同一个三维模型。图 2.2.3 中的两个近似模型都有 2732 个顶点，从图中可以看出算法改进前后生成的近似模型都是高质量的，图中圆球部分并没有进行简化，说明原算法和改进后的算法在具有 2732 个顶点时，对简化边的选择是一致的。图 2.2.4(a)是算法改进后产生的结果，具有 2300 个顶点，图 2.2.4(b)是原算法的结果，也具有 2300 个顶点。这时原算法对模型中圆球部分作了过多的简化，使整个模型看起来比较粗糙，左边的模型其主要的简化部分在模型中圆柱体上，对圆球部分作的简化较少，整个模型看起来更加细致。这说明改进前算法的主要简化对象在球体上，这导致了模型外形较大程度的变化，而改进之后的算法的主要简化对象在中间的圆柱体和灯罩支架上，因为这些对象本身形体较直，多做一些简化并不会引起模型形体过大的变化。

2.2.3 算法的扩展

1、 问题的提出

在虚拟现实及视景仿真的建模过程中，为了使场景更加逼真、形象，使用户对虚拟现实系统有更好的沉浸感，模型的表面都被赋予了一定的属性，如物体的颜色、光照及透明度等等。Michael Garland 的简化算法可以很方便地推广到对具有属性征的表面的简化。在上节中，只考虑了顶点的空间坐标，用 $v=(x,y,z)$ 来表示，同样可以用相似的方法来考虑其属性特征，例如可以用 $c=(r,g,b)$ 来定义一个顶点的颜色坐标，其中 (r,g,b) 分别是顶点的红、绿、蓝颜

色值。具有表面属性的三角面片显然不能用上述的算法处理，我们应该找到一种通用的算法，不仅可以处理几何信息，同样也可以处理属性信息。

2、 问题的解决

(1) 在上节中，只考虑了顶点的空间坐标，用 $v=(x,y,z)$ 来表示它，同样可以用相似的方法来考虑其颜色特征，用 $c=(r,g,b)$ 来定义一个颜色顶点，其中 (r,g,b) 分别是顶点的红，绿，蓝颜色值，可以定义一个误差：$Error(c)=c^TQc$，其中 Q 的求解方法同上节，同样是一个 (4×4) 的矩阵，这样又有了一个基于顶点颜色的误差。但在算法中，只能用一个误差，于是在顶点的坐标误差和颜色误差之间，须做一些选择，比如定义一个新的误差，如下：

$$Error(A)=\alpha Error(v)+\beta Error(c) \tag{2.3.1}$$

其中 α 和 β 分别是基于顶点位置和基于顶点颜色的权值。

这样就有了一个全新的误差，它不仅考虑了顶点的坐标，同样考虑了顶点的颜色，在简化过程中，关于颜色的简化同样可以做到。

(2) 以上是一种思路，另外一种思路是将空间坐标和颜色坐标合在一起，写成一个顶点向量，即 $v=(x,y,z,r,g,b)$，如果是这样，必须扩展上面所定义的误差。

现在我们所用的顶点坐标是一个六元组，是多元空间中的一个点，对于多元空间中的点到其平面的距离，我们作如下考虑：

在多元空间 R^n 中有一个三角形 $T=(p,q,r)$，设 p，q，r 的坐标为 $p=(p_x,p_y,p_z,p_r,p_g,p_b)$，$q=(q_x,q_y,q_z,q_r,q_g,q_b)$，$r=(r_x,r_y,r_z,r_r,r_g,r_b)$。这样，三角形 $T=(p,q,r)$ 所在的平面，可以用两个相互垂直的向量和一个点来表示，如图 2.2.6 所表示：

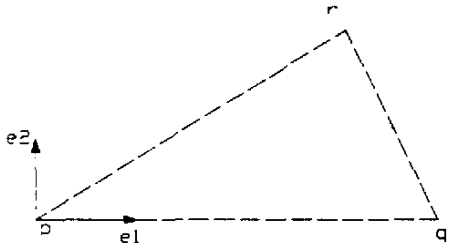


图 2.2.6 三角形 T 的表示形式

图 2.2.6 中

$$e_1 = (q-p) \Big/ \| q-p \| \tag{2.3.2}$$

$$e_2 = \frac{(r-p)-e_1(r-p)e_1}{\|(r-q)-e_1(r-p)e_1\|} \tag{2.3.3}$$

理论上，在多元空间 R^n 中，可以分别求出 $e_1,e_2,e_3\cdots e_n$，$e_1,e_2,e_3\cdots e_n$ 是 n 元空间中相互垂直的单位矢量。

现在求 R^n 空间中任意一点 v 到平面 T 的距离。设 $u=p-v$，则有

$$\| u \|^2 = u^T u = (u^T e_1)^2 + (u^T e_2)^2 + (u^T e_3)^2 + \cdots + (u^T e_n)^2 \tag{2.3.4}$$

(2.3.4)式所表示的实际上是勾股定理的扩展形式。

把上式重写成如下形式：

$$\| u \|^2 - (u^T e_1)^2 - (u^T e_2)^2 = (u^T e_3)^2 + \cdots + (u^T e_n)^2 \tag{2.3.5}$$

注意到式子右边便是 v 到平面 T 的距离的平方。

于是有

$$D^2 = \| u \|^2 - (u^T e_1)^2 - (u^T e_2)^2 = u^T u - (u^T e_1)(u e_1^T) - (u^T e_2)(u e_2^T) \tag{2.3.6}$$

将上式简化，同样用 A,b,c 来表示，则 A,b,c 的值如下：

$$A = I - e_1 e_1^T - e_2 e_2^T \tag{2.3.7}$$

$$b = (p \cdot e_1)e_1 + (p \cdot e_2)e_2 - p \tag{2.3.8}$$

$$c = p \cdot p - (p \cdot e_1)^2 - (p \cdot e_2)^2 \tag{2.3.9}$$

在这里，A 是一个 $(n \times n)$ 的矩阵，b 是一个 n 维向量，c 是一常数。

在计算好 D^2 之后，由 $Q(v) = \sum_{i=1}^{k} \omega_i D_i^2$ 便可求出各个顶点的误差值，简化过程同上节。

在本文中，主要讨论第二种方案。在第一种方案中，虽然也考虑了颜色的作用，但没有考虑颜色与位置坐标之间的相互关系，并且 α 和 β 的取值也难以确定，因为很难找到一个合适的方法来定量的确定颜色与位置坐标之间的关系。在第二种方案中，将颜色和位置坐标放入一个矩阵中，通过矩阵运算，隐式地包含了它们之间的相互关系。最后的实验结果也令人满意，见图 2.2.6。

2.2.4 扩展之后的不足及改进

(1) 基于矩阵计算过程的优化

按照上一节里叙述对算法进行扩展之后，我们需要更多的数据来表示顶点的属性。物体表面的属性越多，计算矩阵中的参数也越多，并且是以二次方的速度增加。这就使计算复杂度令计算机难以承受，所以减少算法的时间复杂度至关重要。

在上一节中，二次方矩阵是 $(3+m) \times (3+m)$ 阶的，其中 3 是指顶点的空间位置坐标 (x, y, z) 三个分量，而 m 是指要描叙物体表面特征所需参数的数目。在矩阵中，非零元素只有 $O(m)$ 个，在做矩阵运算时需要 $O(m^2)$ 的时间。在这一节中，将根据矩阵的相关性质来探讨一种能够减小算法时间复杂度的方法。

注意到矩阵是对称矩阵且是稀疏的，这些性质使我们有更加简单的方法。几何面片的每一个点都是三维空间中的点，如果用 P 来表示这个点，有 $P \in R^3$，由 m 个参数表达的属性是 m 维空间的矢量，如果用 V 来表示此点，有 $V \in R^m$。在此章的第一节中定义了误差的公式为：（假设简化过程为 $(V_1, V_2) \to V$ ）

$$Q(v) = \sum_i \omega_i \cdot Q(v) \tag{2.3.10}$$

其中 i 是与 v 相关联的三角面片的数目，ω_i 是三角面片的面积，$Q(v)$ 是要简化的顶点的二次方误差，即：

$$Q(v) = v^T A v + 2b^T v + c \tag{2.3.11}$$

其中

$$A=\begin{bmatrix} nn^T+\sum_j g_i g_i^T & -g_1 & \cdots & -g_m \\ -g_1 & & & \\ \vdots & & I & \\ -g_m & & & \end{bmatrix} \tag{2.3.12}$$

$$b=\begin{bmatrix} dn+\sum_j d_j g_j \\ -d_1 \\ \vdots \\ -d_m \end{bmatrix} \tag{2.3.13}$$

$$c=d^2+\sum_j d_j^2 \tag{2.3.14}$$

其中 A 是一个 $(3+m)\times(3+m)$ 的对称矩阵，其中前 3 行和前 3 列分别是 $(m\times 3)$ 和 $(3\times m)$ 的矩阵，表示物体的表面特征的参数，余下的 $(m\times m)$ 矩阵是一个单位阵。b 是 $(3+m)$ 的矢量，式中 $d=n^T P_0$，$d_j=g_i^T S_0$，其中 n^T 和 g_i 分别是三维空间坐标的法矢量和属性的法矢量。P_0 是平面中任一点的空间坐标，S_0 是平面中任意一点的属性矢量。

下面把式子化简一下，将 A 写作：

$$A=\begin{pmatrix} C & B \\ B^T & I \end{pmatrix} \tag{2.3.15}$$

按照第一节的做法，对(16)式求一阶导数，并令其等于零，即：

$$\Delta Q^V(V)=2AV+2b=0 \tag{2.3.16}$$

这样就得到

$$AV_{\min}=-b \tag{2.3.17}$$

其中 $V_{\min}$ 是要求的新点，其中包含了空间坐标值和属性参数值。

继续对(2.3.17)式做一些变化：

$$\begin{pmatrix} C & B \\ B^T & \alpha I \end{pmatrix}\begin{pmatrix} P_{\min} \\ S_{\min} \end{pmatrix}=\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{2.3.18}$$

式中 P_m 是要求的顶点坐标中三维空间坐标的矢量，$S_{\min}$ 是属性矢量。

$$b_1=dn+\sum_j d_j g_j \tag{2.3.19}$$

$$b_2 = \begin{pmatrix} -d_1 \\ \vdots \\ -d_m \end{pmatrix} \tag{2.3.20}$$

将(23)中后 m 列乘以 $-\frac{B}{\alpha}$ 再加到前 3 列中，得到

$$\begin{pmatrix} C-\frac{1}{\alpha}BB^T & 0 \\ -\frac{1}{\alpha}BB^T & -B \end{pmatrix}\begin{pmatrix} P_{\min} \\ S_{\min} \end{pmatrix} = \begin{pmatrix} b-\frac{1}{\alpha}Bb_2 \\ b_2 \end{pmatrix} \tag{2.3.21}$$

这样就有

$$(C-\frac{1}{\alpha}BB^T)P_{\min} = b1-\frac{1}{\alpha}Bb_2 \tag{2.3.22}$$

将上式代入(23)式中得到

$$S_{\min} = \frac{1}{\alpha}(b_3 - B^T P_{\min}) \tag{2.3.23}$$

这样改进之生计算 BB^T 只需 $O(m)$ 的时间，计算 $B^T P_{\min}$ 也只需 $O(m)$ 的时间，所以整个算法只需 $O(m)$ 时间就可以完成。实验结果见表 2.2.1。

表 2.2.1：优化前后计算复杂度的比较

模型	原始 模型顶点数	简化后 模型顶点数	原算法的计算时间	优化后算法的计算时间
F16	1393	1000	86，565(ms)	9，758(ms)
Hubble	575	250	86，323(ms)	9，777(ms)

注：本文中数据是在 SGI550，Window2000 环境下的实验数据，实验程序用 VC 和 OpenGL 或 DirectX8.1 SDK 开发

从表 2.2.1 中可以看出，优化后的算法比原算法在计算效率上提高了一个数量级，但近似模型的质量是一样的，见图 2.2.5。

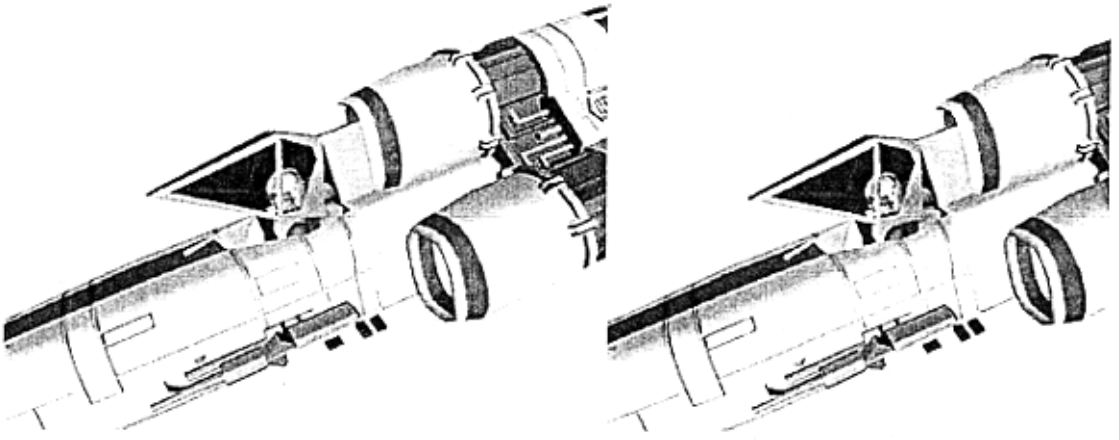


图 2.2.5：算法优化前后的实验结果

图 2.2.5 中左边的图像是原算法的实验结果，有原始模型的 80%的顶点，右边的图像是算法优化后的结果，同样有 80%的顶点，比较两副图像可以发现优化前后的简化质量是一样的。

(2) 子集替代优化

在虚拟现实和视景仿真的场景建模中，很少考虑物体的颜色、顶点法矢等因素。虽然我们需要逼真的效果，但就一般来说，给模型加上逼真的帖图就可以达到我们所需的要求。所以建模时很少对物体的颜色赋值，即使物体有了颜色，在加上贴图或纹理之后，颜色的效果会被贴图遮掉。鉴于此，在这一部分里我们着重讨论物体只有贴图属性的三角面片的简化。

在 2.2.3 节中，将 Garland 算法扩展到了对具有属性面片的简化，在求得新顶点坐标的同时也得到了属性值。如果属性是贴图。则新顶点的贴图坐标也就计算出来了。从理论上讲，由于计算机的固有误差及偶然误差等不良因素的存在，其贴图坐标的计算结果也就不十分精确。这样生成的贴图会产生一些扭曲，并且随着 LOD 层次的增加，这种扭曲会越来越大。既然这样，何不用不经过计算得来的坐标呢？不经过计算的坐标不会产生误差，贴图坐标也就不会产生大的变化。不经过计算的坐标只有原始模型中的顶点坐标，那么我们就只使用这些数据。因为顶点的空间坐标和贴图坐标是同时计算出来的，所以顶点的空间坐标也用能计算了，也不会生成新的顶点，只是不停地合并顶点对。于是算法就从 $(v_1,v_2)\to\bar{v}$ 变成了 $(v_1,v_2)\to v_1$ 或是 $(v_1,v_2)\to v_2$。也就是说在简化边 (v_1,v_2) 时，删去了误差较小的那一点，被删减的那一点的所有剩余边都重新联接到被保留下来的那一点上。这样

就不必计算矩阵的逆、乘和除等运算。无论是时间复杂度还是空间复杂度都大大降低。当然这样一来，对物体原形的会有较大的改变，这是美中不时足的地方。

改进之后的算法流程如下：

(1) 首先计算每个顶点的Q值；

(2) 选择有效的顶点对；

(3) 比较有效顶点对(v_1,v_2)两顶点的误差，选择较小的顶点；

(4) 将误差满足条件的顶点合并，删减误差较小的顶点；

(5) 如简化后的模型满足要求则结束，否则转 2。

图 2.2.6 和图 2.2.7 是实验结果。

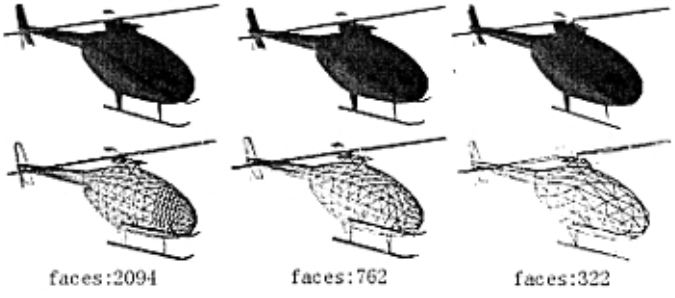


图 2.2.6　简化模型效果

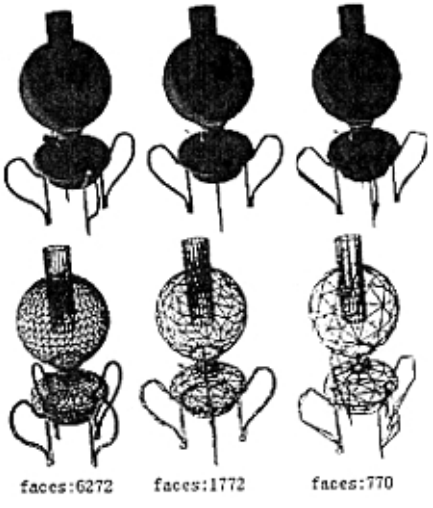


图 2.2.7 简化模型效果

如上所述，将$(v_1,v_2)\rightarrow\bar{v}$改为$(v_1,v_2)\rightarrow v_1$或是$(v_1,v_2)\rightarrow v_2$后，模型的形体会有较大的改变，但在有些领域，这种方法是很有吸引力的，例如递进网格。递进网格中通常使用原始模型的子集来表示简化模型（子集替代），而不计算优化点$\bar{v}$（优化替代）。虽然这样产生的模型质量较差，但可以降低存储空间，因为不需要更多的内存来存储新点$\bar{v}$，也能很好地降低计算复杂度。

2.2.5 总结

本节在 Garland 基本简化算法的基础上，提出了一些改进的方法，首先是 Garland 算法本身的一些缺点。通过改进，保证了在 LOD 层次较大的情况下能更好的保持物体的原形。然后

将算法扩展到对具有属性的面片的简化。并对扩展之后的算法体现出来的高复杂度进行了分析，给出了一种速度非常快的简化方法，通过对计算过程的改进，减小了计算复杂度。最后讨论了只有贴图属性的面片的简化方法。算法在实现的过程中，采取了 Brep 表示的一些思想，所以所有的程序都具有 Brep 的某些特征，如形体的点、线、面是显式表示，能快速地表示物体模型，数据结构复杂，对数据的操作有一定的难度，内存消耗较大等等。

第3章 递进网格

3.1 简介

在计算机图形中，几何模型通常用三角形网格来表示，一个网格包含一个三角形面片集合以及一个顶点坐标集合。为了满足日益增长的计算机图形真实感需要，几何模型变得越来越高度细节化。传统的建模系统一般通过对大量几何模型元素进行各种建模操作来构造细节模型。而这些模型通常必须被剖分成多边形近似网格。然而，不论哪一种情况，这些得到的网格十分复杂，不利于存储、传输和绘制。而且基于计算机图形建立的场景包含大量网格，所以总的复杂度会更庞大。

为了解决这个矛盾，网格简化和多细节层次模型(Level of Details，简称LOD)方法相继产生。这些方法的目的是在保持模型原始形状的同时尽量删除网格中的三角形数目。绘制中，人们首先直接利用简化后的近似网格代替原始网格进行绘制；或者先定义一组不同细节层次的网格模型，再根据不同场景和视点分别进行绘制，如果模型在场景中距离当前视点足够远，系统就用粗糙模型替代精细模型使用，这样可以在视觉上获得良好的效果同时减少场景复杂度。相对网格简化而言，这种LOD思想进一步提高了模型绘制系统的性能。国内外研究人员已提出了多种不同的简化方法[37]，并取得了不少满意的结果。

然而，这种方法只利用了一组离散网格层次，在用一般的网格简化算法构造LOD模型时，通常需要生成多个不同的简化模型，并且为了保证LOD模型间过渡有连续性，而不在模型层次切换是出现跳跃感，常常需要非常多的层次。如果我们要对一组多细节层次模型进行有效的存储、传输和网格压缩就十分困难。H.Hoppe 在文献[12]中介绍的一种递进网格表示方式为这些课题提供了统一的解决方案。这种递进网格把任意网格表示成一个简化网格和一组记录了网格精简信息的序列。简化网格通过一系列顶点分裂操作就可以动态实时地恢复成原始网格。这种方法不仅能够提供一组连续的多细节层次模型，还支持递进方式传输和提供一种有效的网格压缩方式。虽然这种递进网格在不同方面存在局限性，但是它已经被应用于多细节层次领域的进一步研究中。

3.2 递进网格思想及应用

多分辨率分析[98]和递进网格[12]的研究都为物体模型提供了连续的多细节层次描述方式。递进网格提供了一种真正的网格连续分辨率描述，递进网格能够快速有效地生成任意中间层次的网格模型，支持有选择的网格精化操作。递进网格生成的任意两个网格之间存在一种顶点对应关系，这大大方便在网格上建立基于表面映射的关系。

递进网格还是一种出色的网格几何压缩方法，由于相邻两个网格之间的差别就在一次网格操作的结果，并且这个结果相差甚微，递进网格利用这一点可以得到有效的存储方案。由于递进网格描述方法渐进的细化特点，它还可以被广泛应用于网络传输。目前互联网络的应用十分广泛，但是由于数据量庞大，网络带宽有限等因素，一般大数据量图像采用渐进传输的方式；递进网格方式提供了一种图形的渐进传输方法，实现了基于 Web 的几何模型的实时传输。Microsoft 公司已经在 DirectX 5.0 中提供了递进网格的图形接口。

1、可逆的对偶操作

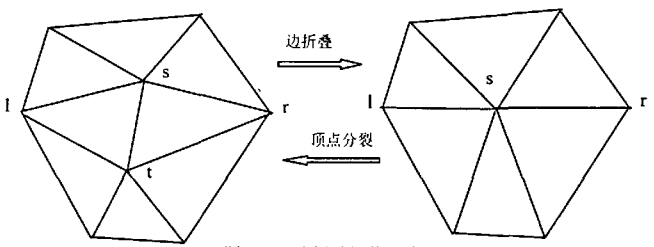


图 3.1：边折叠操作示意

递进网格的基本思想是先在网格简化时同时顺序记录简化操作序列，而根据这些记录对网格进行逆操作就能够完整地恢复原始模型。Hoppe 在文献[12]文中所提出的递进网格建立在以边折叠为简化元操作的网格简化算法基础上。我们可以把基于边折叠的网格简化看成这样一个过程：一个三角形网格 $M_k \in M$ 经过一系列边折叠操作转变成一个相对粗糙的简化网格 $M_0 \in M$ 。这种方法的特点是每次简化迭代所改变的局部网格元素都相同：一条边、与这条边相连的两个顶点和两个三角形，同时因合并这条边而生成一个新的顶点。递进网格因此产生十分规整的信息序列。实际上，递进网格描述方法把原来的网格描述{K，V}，转换成一种新的网格表示方法：一个最简网格模型及一组有序的网格精化操作记录。

图 3.1 例示了三角形网格的边折叠过程。可以看到，一个边折叠操作和一个相应的顶点分裂操作是一组对偶操作：边折叠操作把边 (s,t) 合并成一个顶点{s}，同时删去两个三角形{s， l， t}和{s， t， r}；而顶点分裂操作把顶点{s}分裂为边{s， t}，同时产生两个新的三角形{s， l， t}和{s， t， r}。因此，一组 k 个边折叠操作序列会相应产生一组连续的近似网格序列 M_i：

$$M_k \xrightarrow{ecol_{k-1}} M_{k-1} \xrightarrow{ecol_{k-2}} ...M_1 \xrightarrow{ecol_0} M_0 \tag{3.1}$$

同样，我们还可以利用同一组边分裂操作序列对简化网格进行顶点分裂转换，从最简网格 M_0 获得后续的高细节层次网格，从理论上说这两种操作是一组相互可逆的操作：

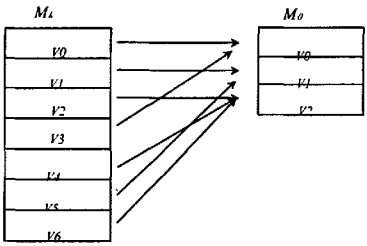


图 3.2：网格边折叠以后顶点的映射关系。

$$M_0 \xrightarrow{vsplit_0} M_1 \xrightarrow{vsplit_1} ...M_{k-1} \xrightarrow{vsplit_{k-1}} M_k \tag{3.2}$$

2、递进网格的构造

递进网格建立在网格简化基础上，目前国内外已经开展了许多关于网格简化算法的研究。这其中有不少网格简化算法基于边折叠元操作[10]。递进网格首先利用网格简化算法生成一个简化模型，这个过程可以预先处理以求获得较好的简化效果。

图 2.3.2 显示了边折叠操作后相应的网格顶点关系变化过程，其中 m_0 为网格 M_0 的顶点数目。因此，某个中间网格 M_i 的顶点集合就可以表示为 $V_i = \{v_0, v_1, \ldots, v_{m_0+i-1}\}$。

当然在实际应用中如何选择折叠边十分重要，这决定了操作记录序列的顺序和简化模型的质量。由于边折叠操作是一个可逆过程，我们可以把它的对偶操作，顶点分裂表示成一个五元组 $vsplit(s,l,r,cv_s,cv_{m0+i})$，其中 cv_s 和 cv_{m0+i} 分别是网格 M_i 的后续网格 M_{i+1} 的 s 点和在这次

操作中新生成顶点 m_n+i 的坐标值。我们定义顶点空间和三维坐标空间分别为 V 和 R^3，那么五元组 (s,l,r,cv_s,cv_{m0+i}) 就是五维信息空间 φ 中的一个点：

$$\varphi = (V,\ V,\ V,\ R^3,\ R^3) \tag{3.3}$$

综上所述，我们可以如下重新理解网格空间中边折叠操作 *ecol* 和顶点分裂操作 *vsplit*，定义映射关系如下：

$$\text{ecol}: \quad (M,\varepsilon) \to (M,\varphi) \tag{3.4}$$

$$\text{vsplit}: \quad (M,\varphi) \to (M,\varepsilon) \tag{3.5}$$

其中，符号 M 和 φ 分别表示为网格空间和边集合空间。*ecol* 是定义在 (M,ε) 空间上，值域为 (M,φ) 的一个映射，即对于原始输入网格 M_i 及其中指定的一条操作边 e，应用法则 *ecol*，得到一个新的网格 M_{i-1} 和一个记录了整个简化过程的五元组 p；同理，vsplit 是定义在 (M,φ) 空间中，值域为 (M,ε) 的一个映射，即对于一个初始网格 M_{i-1} 和一个的五元组点 p，应用法则 vsplit 对 p 中指定的顶点进行顶点分裂操作，恢复一个新的网格 M_i。基于以上关于边折叠和顶点分裂法则，我们知道在一定条件下两者是一对对偶关系，即 ecol 和 vsplit 都是一一映射，对同一输入参数 (m,ε)，$ecol(m,\varepsilon)$ 映射到空间 (M,φ) 中唯一的值；对于参数(m，p)，vsplit(m，p)也得到唯一的值。因此 ecol 和 vsplit 是一组可逆过程，如上图 2.3.1 所示，有如下等价关系成立：

$$\text{ecol}^{-1} \Leftrightarrow \text{vsplit}, \quad \text{即 } \text{vsplit}^{-1} \Leftrightarrow \text{ecol} \tag{3.6}$$

基于上述定义，我们可以重新描述递进网格的概念。Hoppe 定义二元关系(M_0，{$vsplit_0$，$vsplit_1$，...，$vsplit_{k-1}$})为递进网格的描述方式，其中 M_0 为简化后的最简网格；$vsplit_i$ 表示顶点分裂的五元组。即递进网格把普通网格 M_n 重新表示成一个粗糙网格模型 M_0 以及基于这个粗糙模型的一系列有序的顶点分裂信息。这种简捷新颖的表示方法为多边形网格模型的存储管理、绘制和传输提供了一种非常好的解决方案。

3、递进网格的二义性

文献[12]提出的基于边折叠操作的递进网格表示方式(M_0, {$vsplit_0$, $vsplit_1$, ..., $vsplit_{k-1}$})作为一种有效的多细节层次网格模型管理形式，可以十分方便地通过顶点分裂操作序列{$vsplit_0$, $vsplit_1$, ..., $vsplit_{i-1}$}从最简网格 M_0 得到一个任意中间细节层次的网格 M_i。每一次在网格上的顶点分裂操作都会由顶点 s 分裂产生一个新顶点 m_0+i 以及两个新的三角形{s, l, m_0+i}和{s, m_0+i, r}，参见图 2.3.1。假定此处讨论的所有网格的每个三角形的法向量保持一致，一致向外，即三角形的三个顶点顺序一致。在一般情况下，根据五元组(s, l, r, cv_s, cv_{m_0+i})进行的顶点分裂操作过程为：首先在输入网格 M_i 中寻找所有与分裂母点 s 相邻的三角形，并且根据已知分裂母点 s 的左右邻接顶点 l 和 r，改变所有邻接三角形由于顶点而引起的连接关系变化，其中一部分原先与顶点 s 相连的三角形将与新生成的顶点 m_0+i 相连。那么哪些三角形需要改变呢？由于事先存在对网格类型的限制，我们只要在顶点 s 的邻接三角形中从左起始点 l 开始沿逆时针方向搜索到右终点 r 为止之间的三角形就可以了。在实验中，很多网格数据都可以顺利地应用这种方法。

但是，在某些特殊的网格中这种判断方法碰到一些困难。我们来考虑一个特殊网格 Ma_i 的一次边折叠操作，如图 3.3a 示，根据我们在公式(3.4)关于网格边折叠操作的映射定义，我们可以把 Ma_i 中边{m, n}合并成顶点 m 的这个边折叠操作表示成如下形式：

$$ecol(Ma_i, \{m, n\}) = (M_{i-1}, (m, l, r, cv_m, cv_n)) \tag{3.7}$$

现在我们再来观察另外一个网格，如图 2.3b 示。同理，根据公式(3.4)中的定义，我们也可以把网格 Mb_i 中边{m, n}合并成顶点{m}这样一个边折叠操作表示成如下形式：

$$ecol(Mb_i, \{m, n\}) = (M_{i-1}, (m, l, r, cv_m, cv_n)) \tag{3.8}$$

从(3.7)和(3.8)我们可以看到，两个表达式完全一样。而这两个公式是根据两个不同的网格来描述的。因此这种情况反映出来在一定条件下映射 ecol 和 vsplit 并不是一一映射，即他们并不是一个真正的可逆过程。

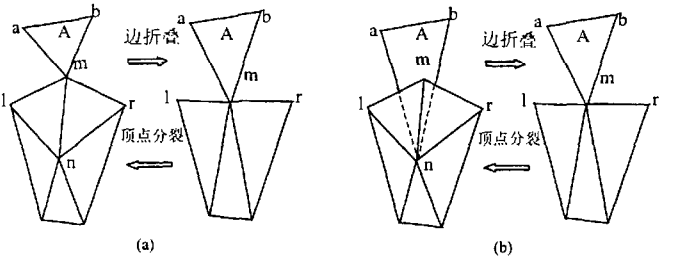


图 3.3：顶点分裂二义性示意

在图 3.3 所列两个网格模型的边折叠操作中两者都得到了相同的结果，所以反过来，当简化网格 M_{i-1} 的顶点 m 分裂时，边折叠过程产生的五元组(m，l，r，cv_m，cv_n)会使分裂映射 vsplit 产生二义性：这个网格 M_{i-1} 既可以转换到图中(a)的网格模型；也可以转换到(b)的情况。实际上，由于原先算法要求根据三角形面片法向量寻找分裂母点 m 的邻接顶点，那么在图 3.3a 和图 3.3b 中，由于三角形{a，b，m}与其他顶点 *m* 的邻接三角形不相连，所以计算过程就不清楚该三角形在简化操作前是属于顶点 *m* 还是顶点 *n* 的邻接三角形。也即算法无法判断三角形 *A* 的顶点 *m* 是否应当改为顶点 *n*。虽然这种类型的网格可能不会在初始网格中出现，但是有可能经过多次网格转换后出现。

由于映射 vsplit 存在上述二义性，网格 M_{i-1} 经过一系列顶点分裂操作后可能产生不同于原始网格 M_k 的最终网格，甚至改变网格模型的拓扑结构。事实上，实际网格会存在类似的局部区域。而如果要避免出现类似图 3.3b 的结果，那么就必须进行复杂的合法性检测，这样会引起算法整体效率的下降。

如果想简单地解决上述二义性问题，我们可以在网格简化时直接把每次顶点分裂所需要的邻接三角形信息全部记录下来。某些多细节层次方法利用了类似的方法[33]。但是因为实际网格中每个顶点的邻接三角形数目不确定，而且随着网格简化过程的进行，新生顶点的邻接三角形数目会变得越来越大，比如我们在简化一个 Bunny 模型的时候发现某些顶点的邻接三角形数目有上百个。所以这种递进网格结构会十分无序和复杂，同时给数据结构设计也带来了麻烦。

其实不仅仅因为原先的递进网格存在这种二义性情况，在某些特殊网格的操作中它同样不能适用。根据我们前面的介绍，递进网格要求操作网格的类型为法向量一致的规则网格，这样才可以完全恢复。

3.3 一种快速生成的递进网格

在文献[35]中，给出一种 PM 的数据结构，如下表所示：

```
struct MRMFace
{
    INT ObjectID;
    INT LODFaceID;
    float bError;  //出生误差
    float dError;  //死亡误差
};
```

此结构是一线性表，记录了模型中所有三角面片的状态，包括原模型的三角面片及简化生成的三角面片。第一项 FaceID 是三三角面片的 ID 号，也就是三角面片的编号；BError 表示三角面片生成时的当前二次方误差，称为出生误差；DError 是三角面片被简化时的当前二次方误差，称为死亡误差。从 BError 到 Derror 这个区间称为三角面片的生存期，只有当误差 E 在某个三角面片的生存期内时，此三角面片才会被显示出来。此结构对已经经过计算的 LOD 层次的模型显示非常快。例如，当前的 LOD 层次为 3，前 3 个层次（0，1，2）都已计算过，原模型的三角面片和新生成的三角面片都被添加到线性表中，如果要显示前 3 层的模型，程序只需搜索一下线性表即可得到需要显示的三角面片，无需再做其它运算，这也是增加空间复杂度来减小时间复杂度的一种方法。

此算法虽然快速，但经过实验发现它存在一定的问题，见实验数据如下：

表： 3.1 数据结构修改前后，不同 LOD 层次显示速度的比较

模型名称	算法	顶点数	LOD 层次	速度(FPS)	LOD 层次	速度(FPS)
Puppy	原算法	69451	0	52	3	76
	改进后算法	69451	0	53	3	356
Foot	原算法	4204	0	732	3	1054
	改进后算法	4204	0	732	3	5421

从表中可以看到原数据结构表示的模型在不同的 LOD 层次下，其显示速度是基本不变的，或是变化不大，通过分析，我们发现了问题所在。

在原数据结构之下，LOD 层次越高，其线性表中的数据量越大（即添加进线性表中的三角面片越多），线性表的容量增长是很快的，这一点可以从下面的表中得知：

表： 3.2 不同LOD层次之间三角面片数目的比较(puppy模型)

LOD层次	0层	1层	2层	3层
三角面片数目	69451	47570	20454	8494

线性表容量的不断增加，使程序在搜索线性表时所花费的时间也增加，多做了无用的判断，于是显示速度有所下降。对于越简单的模型其显示越慢，这当然不是我们所期望的。通过修改数据结构，可以克服这一缺点。修改后的数据结构如下：

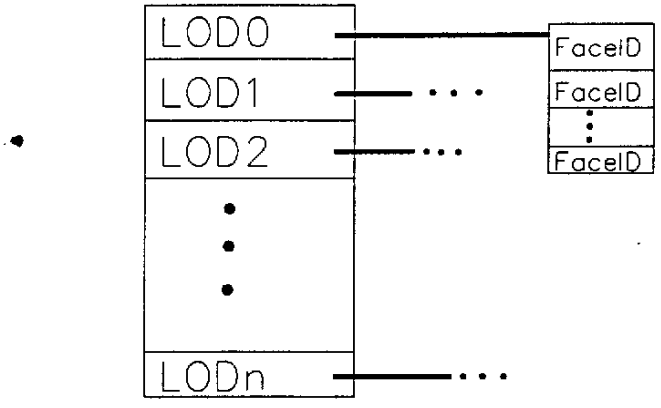


图3.4 修改后的数据结构

改进后的数据结构加入了一个表头，表头中的每个元素都是一个指针，指向当前LOD层次模型的线性表的地址，原数据结构线性表中的出生误差和死亡误差就不必存在了，因为每一个FaceID都用索引到与之相关的信息。这样，程序在显示某一个LOD层次时，不用搜索所有的数据项，不必再做无用的判断，从而提高了显示速度，并且对于LOD层次越高的模型，其显示速度明显高于LOD层次低的模型，数据可以从表3.1中看出。

3.4 网格密度及应用

1、概念

计算机所做的一切事情都是为用户服务的，我们总是希望它能够非常聪明的合理的做事，与用户的交互越少越好，计算机做事越智能越好，这样就为程序设计员提出了更高的要求。

在前面所述的简化算法当中，需要我们人工参与的事情有很多，例如，在建模时，模型总是做的有大有小，其精细程度也有所不同，递进网格怎样来确定LOD层次，计算机是不能自己确定的，必须有人工的参与，但我们所规定的未必符合客观的规律，所以需要经过多次的

实验才能最后确定下来，这就给我们带来了诸多的不便。这一节主要给出一个全新的概念，通过它来确定 LOD 的层次，使程序能自动地选择层次，减少人工的干预，并符合客观规律。

网格密度：指物体模型与视点的距离一定时，形成模型所有三角面片的某一度量的平均值。

在本文中，此度量可以取三角面片的面积或是周长，只要这个量可以表示此时模型的精确程度。有了网格密度这一概念，就可以通过几何关系来确定 LOD 层次。

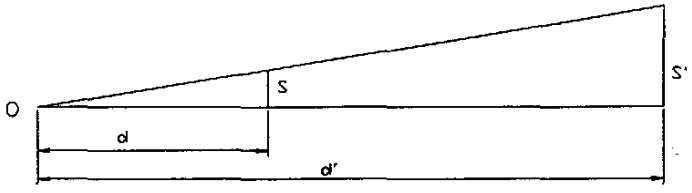


图 3.5 网格密度的确定

如图 3.5 所示，假设一开始，某一面片距视点的距离为 d，此时面片的面积为 s，物体不断远离视点，当距视点距离为 d'时，面积为 s'。因为 s 是原始物体与视点的距离，显示平面在 s 的附近。根据假设 s 是网格精确度的一种度量，那么，当视点距物体 d'是，s'也应该是用户感到满意的。这样，在模型距视点的距离为 d'时，模型的简化程程序应该将物体模型简化到网格密度为 s' 时的模型。根据几何关系，可以很容易地求出 s' 的值，$S'=\frac{d's}{d}$，有了这些关系，可以控制简化程度，改进后的流程如下：

(1) 初始化，确定原始网格密度 s；

(2) 当视点拉远之后，计算当前网格密度 s'；

(3) 简化模型；

(4) 如 $S'<\frac{d's}{d}$，转 2，否则简化完成，显示模型。

2、性质分析

网格密度的优缺点：网格密度为 LOD 的自动化提供了一条便捷的方法，此方法简单便于实现，计算复杂度低，能够和 LOD 简化算法很好地结合使用。但此方法对原始模型的设计要求较高，它需要在建模时便确定模型的精确度，如果简化不令人满意，需要重新修改原始模型。

网格密度提出之后，LOD 层次的概念有了一定的变化，从上面的叙述中，我们知道 LOD

层次是离散的，并且是跳跃式的，层次之间相隔一定的距离，层次之间的变化是突然的。但用网格密度控制的简化过程是根据距离而定的，距离变化大，简化程度就在大，距离变化小，简化程度也小。一个极端的情况是，视点的变化是连续的，那么简化层次也就变成连续的。所以此节中，LOD 层次这个概念被 LOD 简化程度来代替更加贴切一些。

3、 网格密度所解决的问题

(1) 离散多分辨率模型(Discrete Multi-resolution Model)

建立多分辨率模型简单的方法是生成一组越来越简单的模型，本文的前几节正是讨论的这个问题。对于这种方法，给定某一帧，程序只需选择合适的简化模型，这种模型称为离散多分辨率模型，如图 3.6 所示。

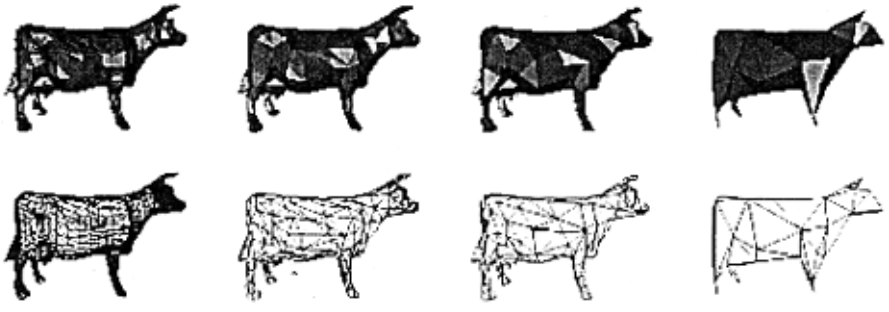


图 3.6 离散多分辨率模型

图中所示的模型有 4 个层次，程序只能选择此四个模型中的某一个用来显示，对于层次中间的模型，离散多分辨模型是无能为力的。离散多分辨率模型表示包含一组不同层次的模型，通过某种阈值来控制层次间的切换，此模型的主要特点是简单，便于实现，缺点是运行时，有效的细节层次是有限的，如果用户需要某一中间层次的模型，由于此层次模型在离散多分辨率模型中并不存在，所以程序只能选择此层次的上一层次模型或是下一层次模型，于是显示给用户的或过于繁杂，或过于简单。即使如此，离散多分辨率模型还是有着广泛的应用，如果某一物体的表面都严格地限制在同一范围内，则离散多分辨率模型是控制 LOD 的有效手断。许多漫游系统都采用了离散多分辨率模型[19]，包括 RenderMan[40]、IRIS Performan[41]。

(2) 网格密度对离散多分辨率的改进

正如上面所说，离散多分辨率模型在某种环境下是非常有效的，但在另外一些环境下其功能是不足的，网格密度的提出给出了一个扩展其功能的途径。通过网格密度对简化程度的

控制，用户可以观察任意某个层次，这个层次上的模型是经过严格定义的，也就是说，网格密度提供了一种生成准连续多分辨率模型的方法。

用网格密度的概念可以很容易地产生任意 LOD 层次模型，可以在一定程度上解决离散多分辨率模型的不足。从网格密度的计算过程中可以看到，我们用物体到视点的距离来控制 LOD，给定一个距离便有一个特定的 LOD 层次与之对应，反之，一个给定的 LOD 层次也只有一个特定的距离与之对应，即 LOD 层次与物体到视点的距离是一一对应的。用户要观察某一中间层的模型，只需定义一个距离即可。例如，LOD 层次如下：

$$M_0 \to M_1 \to M_2 \to \cdots \to M_k$$

假设下一层次保留上一层次 80%的顶点数。现在用户需要观察 M_1 与 M_2 的中间层次 M'，M' 层中保留了 M_1 层中 90%的顶点，则 M' 中保留了原始模型中 82%的顶点数。如果按照传统的 LOD 方法，用户观察 M' 层次需要在任二个层次之间加入一个中间层，LOD 层次需要改动如下：

$$M_0 \to M_1' \to M_1 \to M_2' \to M_2 \to \cdots \to M_k$$

在上面的层次中，下一层次保留的顶点数是上一层次的 90%。这样便增加了层次数目，对于多分辨率模型来说，这样处理要增加近一倍的存储空间，也增加了计算时间。

使用网格密度来控制 LOD 便不需要改变 LOD 层次。如上面的例子中，M' 层次中视点与物体的距离可以通过与用户的交互得到。如层次 M_1 中，物体与视点的距离为 10，M_2 中物体与视点的距离为 20。一开始，与 M' 相对应的距离 d 可以取为 15，然后观察 M' 层次的顶点数是否是原始模型的 82%，如果不是，则对 d 微调，直到简化模型满足条件。这样，用户与计算机交互几次，便可以很容易地得到 M' 层次的 d 值。

(3) 连续多分辨率模型(Continuous Multi-resolution Model)

正如上节所述，离散多分辨率模型有着先天的不足，解决这个问题的方法就是连续多分辨率模型。连续多分辨率模型的典型例子是地形(Terrain)，地形是一块巨大的表面，表面上有表示地貌的凸起或凹陷。视点在地形的正上方，按照多分辨率模型的概念，离视点近的地方分辨率高，离视点远的地方分辨率低，这样在一个物体模型中便产生了多个连续的细节层次，这样的模型称为连续的多分辨率模型[42]。

连续多分辨率最早是在飞行模拟系统中提出的，至今已有 20 多年的历史，20 年中，国内外学者提出了许多的连续多分辨率模型简化方法[43][44]，大多数的简化方法都是基于对地形

表面规则的细分，如四叉树或是八叉树，但基于通用三角面片的连续多分辨率模型的研究还不多。文献[45][46]提出了一个很好的三角面片的通用简化算法。此算法使用了顶点层次的概念。顶点层次作为一种特殊的多分辨率模型表示引起了当代很多学者的注意，它的一个主要特点是可以产生标准简化层次的一个副层次。

(4) 网格密度在离散多分辨率的应用

上一节提出的网格密度的概念可以很好地用于连续多分辨率模型中。地形中不同区域的细节层次与距离视点的远近程度有关，而对网格密度的控制也采用了与视点的距离，这样网格刻度无需作较大的修改便可以应用于连续多分辨率模型。唯一改动的是将视点与物体的距离变为视点与三角面片的距离。

对于连续多分辨率模型中的每一个三角面片都可以确定一个 LOD 层次，距视点近的三角形 LOD 简化程度小，距视点远的三角形 LOD 简化程度大。从视点开始向四周到无限远（模型边缘），通过网格密度检测每一个三角形的简化程度，如果某个三角形的简化程度相对于与视点的距离太小，则从三角形的三条边中取一个误差最小的边折叠，然后更新数据，再次检测周围三角形的简化程度，如不满足条件则再次查找误差最小的边折叠，直到三角形的简化程度适合与视点的距离。

如果按照上述的算法处理地形模型，其复杂度是很高的，因为算法要重复检测模型中每一个三角形的数据。为简化计算，可以将与视点距离在某一范围的三角形视为同一层次。例如，可以将距视点 0~10 的三角形视为一个层次，将距视点 10~20 的三角形视为另一层次，如此类推。这样算法无需检测模型中的每个三角形，减少了计算时间。实验结果如图 3.7：

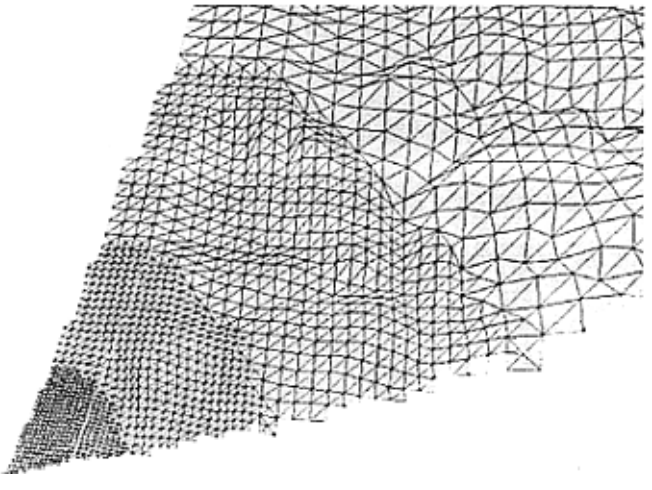


图 3.7 用网格密度定义的地形 LOD 层次

4、带预处理的自动 LOD 技术

在 VR 或视景仿真中，场景都包含了大量的多边形，少的有几百万，多则上千万上亿，这此庞大的数据对计算机来说是个沉重的负担，并且在 LOD 简化过程中有大量的浮点运算，对这些多边形的运算使计算机力不从心，这样在模型的显示方面会有些不尽人意的地方，本节给出一种预处理的思想克服这种不足。

在层次的转换过程中有些停滞会使用户不满意，但在程序的初始化过程延长 10 秒 20 秒，用户还是可以接受的，基于这个原因，我们可以在程序的开始时就计算前 n 个层次的数据并保存在数据结构中，这样在层次的转换时不需要通过计算便可以完成，不会形成停滞。

N 的取值不过大，实验表明取 3-5 之间最为合适。在上一节图表中的数据表示简化一个层次会删减大量的三角面片，三五个层次之后，三角面片的数目大大减少，这时计算机完全可以完成简化任务面不会发生停滞现象。

3.5 总结

这一节主要研究了递进网格技术，完成的工作有：递进网格的生成，对原数据结构的改进，提出了网格密度的概念和带预处理的 LOD 技术，这些方法的提出解决了现有算法中的一些缺憾，主要体现在两个方面：

1 无二义性：

本节所介绍的算法详细记录了简化前后各个层次模型每个三角形信息、边信息和顶点信息，恢复时不会出现二义性，当然这是以存储空间为代价的。

2 快速恢复：

正如上面所述，算法记录了各个层次的所有点、线和面的信息，恢复时没有复杂的计算。程序只需要将后备缓冲中的数据填充到当前的帧缓冲中即可，体现出恢复快速的特点。

第 4 章 对 BSP 树的一些研究

4.1 简介

BSP 树是 Binary Space Partitioning Tree 的缩写，它是一种标准二叉树，是用于在一个空间中组织物体对象的数据结构，一般用来在 n 维空间中进行对象(polytopes)排序和搜索，在计算机图形学领域中经常用于而后消隐的光线跟踪。整个 BSP 树表示整个场景，其中每个树节点分别表示一个凸子空间。每个节点里面包含一个“超平面(Hyperplane)”作为二分空间的分割平面，该节点的两个子节点分别表示被分割成的两个子空间。另外每个节点还可以包含一个或者多个几何对象(polytopes)。

BSP 树在 2 维和三维空间中的应用比较普遍并且容易被理解，但是 BSP 树结构的定义能够表示更高维空间的场景分割。很显然，在 2 维和三维空间中，BSP 树节点中保存的对象(polytopes)分别为线段和多面体。

BSP 树结构可以通过递归和层次的方式把 n 维空间细分成凸子空间(convex subspaces)，BSP 是通过“超平面”递归地分割当前子空间构造起来的。在 n 维空间里的一个“超平面”就是一个 n-1 维的对象体，例如，在三维空间里的“超平面”就是一个平面；二维空间里的分割“超平面”就是一条直线。

由于 BSP 树是一种非常高效的排序和分类数据结构，它的应用非常广泛，包括隐藏面消除、光线跟踪层次结构、实体造型和机器人运动策划等

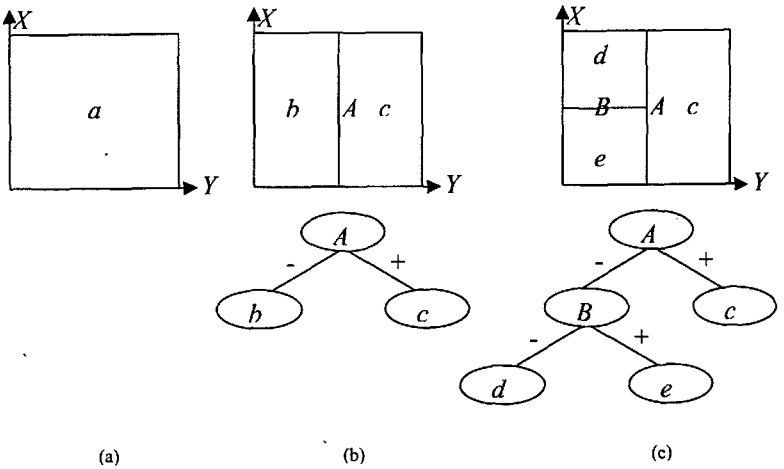
为了说明简单，下面我们以 2 维空间为例描述 BSP 树的原理和构造，并且假定只处理平行于 X 和 Y 轴的线段对象。

给定一个 2 维空间 X-Y，如图 2.4.1 所示，首先选择第一次空间分割的分割“超平面”，这里选择为一条平行于 X 轴的直线作为初始 BSP 树的根节点，如图 2.4.1(b)所示。以后每次分割都在各自的子空间里选择一条合适的分割线，不断地把子空间分割成更小的空间，直到满足一定的收敛条件。最后产生一个被“超平面”细分的空间并且生成一棵表示整个场景空间的 BSP 树。

BSP 树是一种空间分割层次结构，与空间四叉树和八叉树十分类似，但是四叉树和八叉树一般选择平行于坐标轴的分割平面，每次把一个空间均匀划分成四个或者八个子空间，而

BSP 树不仅可以模拟四叉树和八叉树的划分方式，还能够任意选择划分的“超平面”，把整个空间分成若干不规则的子空间，有较大的灵活性。

BSP 树结构中每个节点的两个子节点分别表示背向“超平面”法向量方向的子空间和正



(a)　　(b)　　(c)

图 4.1：2 维空间 BSP 树分割及 BSP 树的构造过程。

对“超平面”法向量方向的子空间，节点同时还包含了该子空间内的多面体对象。三维的 BSP 树的建立和二维的 BSP 树的建立相类似，如图 4.2 所示。

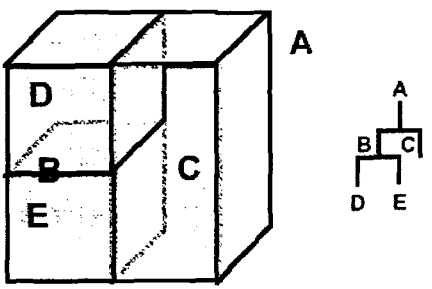


图 2.4.2 三维 BSP 树的建立

2.4.2 BSP树与多细节层次模型

在本文前几章中分别详细介绍了多细节层次模型的自动简化方法、动态多细节层次模型的实时生成方法及其应用。这些多边形模型最后输出到绘制系统，还要经过模型绘制阶段，包括场景裁剪、不可见物体剔除、消隐和光照等，BSP树可以用来帮助处理隐藏面消除。BSP树具有视点无关性这个特点，许多图形系统都支持BSP树结构。

BSP树在多细节层次中的应用很明显，在将空间分割之后，离视点近的子空间中采用较高细节层次，离视点远的子空间中使用较低细节的层次。

如果场景为动态，存在模型的增减，那原先生成的BSP就不合适了，一般的改进方法为用BSP树逻辑操作修改BSP树结构，例如场景内有移动的物体这种情况。

存在多细节层次模型的场景实际是一种动态场景。随着视点的改变，应用更换相应的LOD模型，从而改变了整个场景，需要重建BSP树。如果系统应用静态LOD模型，也即原先计算物体若干个层次的模型，那么可以同时生成各自的BSP树，使用某LOD模型时就调用相应的BSP树绘制，Wiley在文献[47]中利用这种方法处理地形数据和CAD模型取得比较好的结果。

2.4.3 BSP树的创建及绘制

1、BSP树的构造算法

在三维空间中给定一组三角面（可以是多边形，但本文所有的算法都是针对于三角面片的，所以在这里只讨论三角面），我们要创建一个BSP树，树中包含所有的三角面，算法一般包括三步：(1)选择一个分割面；(2)用分割面分割三角面集，生成两个子集；(3)递归分割两个子集，直到满足条件。

分割面的选择需要考虑BSP树将怎样被使用以及排序的效率标准。对于某些情况，经常采用用户输入的面，而在另外一些情况，我们需要估计分割面的选择是否有效，或是选择一个*平衡树，平衡树中的每个子结点都包含相同数目的三角面。当然实现平衡树是需要一定代价的。分割三角面集的过程是根据分割平面归类三角面的过程。*如果一个三角面完全归于分割平面的一边或是另一边，则它不需要修改，把它加入两个子集中的一个即可。如果三角面跨过了分割平面，则需要将三角面分成两部分，这两部分需要加入到相应的子集中。分割过程的停止条件依赖于特定的应用目的，在一些情况下，通过规定子结点中包含三角面的最

大数目来达到此目的，还有的方法通过规定树的最大深度来确定。

下面是BSP树的结构

```
struct  BSP_tree
{
   plane     partition;  //分割平面
   list      polygons; //三角面集合
   BSP_tree  *front, *back; //前后BSP树
};
```

创建BSP树的代码如下

```
void    Build_BSP_Tree (BSP_tree *tree, list polygons)
{
   polygon   *root = polygons.Get_From_List ();
   tree->partition = root->Get_Plane ();
   tree->polygons.Add_To_List (root);
   list      front_list,
             back_list;
   polygon   *poly;
   while ((poly = polygons.Get_From_List ()) != 0)
   {
      int   result = tree->partition.Classify_Polygon (poly);
      switch (result)
      {
         case COINCIDENT:
            tree->polygons.Add_To_List (poly);
            break;
         case IN_BACK_OF:
            backlist.Add_To_List (poly);
            break;
         case IN_FRONT_OF:
            frontlist.Add_To_List (poly);
            break;
         case SPANNING:
            polygon   *front_piece, *back_piece;
            Split_Polygon (poly, tree->partition, front_piece, back_piece);
            backlist.Add_To_List (back_piece);
```

```
            frontlist.Add_To_List (front_piece);
            break;
        }
    }
    if ( ! front_list.Is_Empty_List ())
    {
        tree->front = new BSP_tree;
        Build_BSP_Tree (tree->front, front_list);
    }
    if ( ! back_list.Is_Empty_List ())
    {
        tree->back = new BSP_tree;
        Build_BSP_Tree (tree->back, back_list);
    }
}
```

2、分割过程的实现

空间的分割过程是确定一个三角面属于分割平面的哪一侧的问题，这个过程可以通过前-后检测来实现。如果三角面的每一个点都在分割平面的一侧，则整个物体就在这一侧，这时物体不需要进一步分割；如果有些点在平面的另一侧，即物体跨过了分割平面，则需要进一步将物体分割。判断点在平面哪一侧的过程是这样的：将点的三维空间坐标代入到平面方程中，如方程大于零则点在分割平面的前侧，如方程小于零则点在分割平面的后侧，如等于零则在平面上。

空间的分割代码如下

```
void Split_Polygon (polygon *poly, plane *part, polygon *&front, polygon *&back)
{
  int   count = poly->NumVertices (),
        out_c = 0, in_c = 0;
  point ptA, ptB,
        outpts[MAXPTS],
        inpts[MAXPTS];
  real sideA, sideB;
  ptA = poly->Vertex (count - 1);
  sideA = part->Classify_Point (ptA);
  for (short i = -1; ++i < count;)
```

```
    {
        ptB = poly->Vertex (i);
        sideB = part->Classify_Point (ptB);
        if (sideB > 0)
        {
            if (sideA < 0)
            {
                //计算交集
                vector v = ptB - ptA;
                real   sect = - part->Classify_Point (ptA) / (part->Normal () | v);
                outpts[out_c++] = inpts[in_c++] = ptA + (v * sect);
            }
            outpts[out_c++] = ptB;
        }
        else if (sideB < 0)
        {
            if (sideA > 0)
            {
                // 计算交集
                vector v = ptB - ptA;
                real   sect = - part->Classify_Point (ptA) / (part->Normal () | v);
                outpts[out_c++] = inpts[in_c++] = ptA + (v * sect);
            }
            inpts[in_c++] = ptB;
        }
        else
            outpts[out_c++] = inpts[in_c++] = ptB;
        ptA = ptB;
        sideA = sideB;
    }
    front = new polygon (outpts, out_c);
    back = new polygon (inpts, in_c);
}
```

3、BSP树的绘制

BSP树的绘制可以通过递归来实现，

```
void  Draw_MRBSP_Tree(MRBSP_TREE *tree, point viewpoint, LEVEL t)
{
   if (tree->Empty())    //达到收敛条件，返回
       return;
    //判断视点与当前节点分割平面的相对位置
   switch (tree->partitionPlane.Classify_point(viewpoint)){
    //视点位于分割平面前侧，先处理背面的子空间
   case FRONT:       Draw_MRBSP_Tree(tree->backSpace, viewpoint, t);
                     //判断分辨率层次
                     if (tree->polygons.ResolutionLevelIs(t))
                           tree->polygons.Draw_Polygon_List();
                     Draw_MRBSP_Tree(tree->frontSpace, viewpoint, t);
                     break;
     //视点位于分割平面后侧，先处理前面的子空间
   case BACK:        Draw_MRBSP_Tree( tree->frontSpace, viewpoint, t);
       //判断分辨率层次
                     if (tree->polygons.ResolutionLevelIs(t))
                           tree->polygons.Draw_Polygon_List();
                     Draw_MRBSP_Tree(tree->backSpace,viewpoint, t);
                     break;
     //视点位于分割平面内，则任意选取一个子空间处理
   case INSIDE:      Draw_MRBSP_Tree(tree->backSpace, viewpoint, t);
                     Draw_MRBSP_Tree(tree->frontSpace, viewpoint, t);
                     break;
}
 return;
}
```

2.4.4 实验结果

BSP树的实验结果如图4.3所示。

图4.3 BSP树的场景绘制结果

2.4.5 总结

本节介绍了BSP树的基本概念，基本特点，BSP树与多分辨率模型的关系，分析了BSP树的数据结构和实现算法，最后简单介绍了BSP树的绘制方法并给出了实验结果。

第5章 后继的工作

前几章主要讨论了三维模型的简化方法、递进网格的思想及算法，最后讨论了BSP树的基本概念。这些内容组成了模型简化的主体，是任何LOD模型所必需考虑的问题，但有这一些还是不足的，国内外的学者还提出了许多能够减少复杂度的方法。

在论文的编写过程中，有一条主线贯穿其中：本文放弃了算法的空间复杂度，以便尽可能地减小时间复杂度。这样一来，所有的算法都导致了物理空间的极剧膨胀，所以在后继的工作中，第一个是解决的问题就是数据的压缩问题。

3.1 数据压缩

本文所提出的所有算法都有一个共同的特点，即舍去了算法的空间复杂度，全力提高算法时间效率。对于大型的模型而言，包含的顶点数目异常庞大，这时必须考虑算法的空间复杂度。国内外学者也提出了诸多的LOD数据压缩算法[19]。

3.2 视点相关

在提高模型显示速度的方面，还有一个很热门的课题，就是视点相关。其实在场景绘制中，选择合适的细节模型不仅取决于对象的远近，还应当包括观察视区，可见性，以及其他众多视觉因素。视点相关的LOD算法目前也有好多，但大多是基于三角形删减的简化算法。本文所讨论的边折叠算法也可以进一步优化，将视点的位置变化考虑进去，形成视点相关的LOD算法。下面是一个视点相关的LOD模型说明，见图3.2.1。

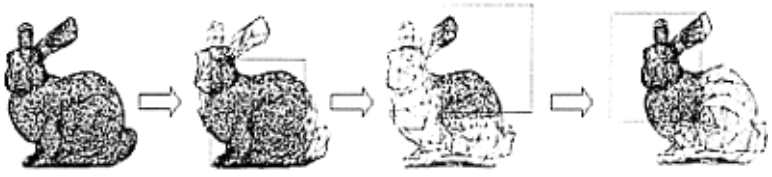


图3.2.1 一种视点相关的LOD模型

3.3 两种多分辨率模型的混合编程

在一个视景仿真场景模型中，存在两种多分辨率模型，即离散多分辨率模型和连续多分辨率模型，两种模型的简化过程是不一样的，在程序的设计过程中必须区别对待。使两种不同分辨率模型的简化过程协同工作的最好方法是多线程，一个线程控制离散多分辨率模型另一线程处理连续多分辨率模型。

致 谢

本文是在我的导师郑永果教授的悉心指导下完成的。从论文的选题、研究思路的确定、论文的撰写直到论文修改的整个过程中，郑老师倾注了大量的心血和精力。在我三年的硕士研究生学习生活期间，我得到了郑老师的悉心的教诲和无微不至的关怀，使我在生活和学习上受益非浅。郑老师认真求实的作风和积极向上的人生态度永远是我做人和做事的坐标和典范，在此谨向他表示最衷心的感谢和崇高的敬意。

在我三年的学习生活中，信息科学与工程学院和研究生处的各位领导及老师也给予了我很大的帮助，在此表示深深的谢意！还要感谢三年来一同学习和生活的诸位同学的关心和帮助，你们的真诚和友情将是我一生的财富。

最后感谢在山东科技大学学习生活中给予我关心、爱护的所有老师、同学！

参考文献

[1] M.E.Algorri and F.Schmitt. Mesh Simplification. In Proceedings of EUROGRAPH'96, 1996: 77-86.

[2] D.Aliaga, J.Cohen, A.Wilson, H.Zhang, C.Erikson, K.Hoff, T.Hudson, W.Stuerzlinger, E.Baker, R.Bastos, M.Whitton, F.Brooks, and D.Manocha. A Framework for the Real-Time Walkthrough of Massive Models. UNC TR: 98-013, 1998

[3] G.Barequet, B.Chazelle, L.J.Guibas, J.S.B.Mitchell and A.Tal. BOXTREE: A Hierarchical Representation for Surfaces in 3D, In Proceedings of EUROGRAPHICS'96, 1996:387-396

[4] S.E.Chen. Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System, In Proceedings of SIGGRAPH'90, 1990:135-144

[5] P.Cignoni, E.Puppo, R.Scopogno Representation and Visualization of Terrain Surfaces at Variable Resolution, In Scientific Visualization'95, World Scientific, 1995

[6] P.Cignoni, C.Montani, C.Rocchini, R.Scopogno, Resolution Modeling. Technical Report C97-02, CNUCE-C.N.R., Pisa, Italy, 1997

[7] J.H. Clark, Hierarchical geometric models for visible surface algorithms, Communications of the ACM, 19(1976):547-554

[8] J.Cohen, A.Varshney, D.Manocha,G.Turk, H.Weber, P.Agarwal, F.Brooks, and W.Wright. Simplification Envelopes, In Proceedings of SIGGRAPH'96,1996:317-345

[9] J.Cohen, M.Olano, and D.Manocha. Appearance-Preserving Simplification. In Proceedings of SIGGRAPH'98, 1998

[10] H.hoppe, T.DeRose, T.Duchamp, J.McDonald,and W.Stuetzle. Mesh optimization. In Proceedings of SIGGRAPH'93, 1993: 19-26.

[11] H.Hoppe, T.DeRose, T.Duchamp, M.Halstead, H.Jin, J.McDonald, J.Schweitzer, and W.Stuetzle. Piecewise smooth surface reconstruction. In Proceedings of SIGGRAPH'94, 1994: 295-302.

[12] H.Hoppe. Progress Meshes. In Proceedings of SIGGRAPH'96, 1996: 99-108.

[13] H.Hoppe. View-Dependent Refinement of Progressive Meshes, Computer Graphics, SIGGRAPH'97 Proceedings:189-198

[14] H.Hoppe. Efficient Implementation of Progressive Meshes. Technical Report, MSR-TR-98-02, 1998

[15] M.Garland, P.S. Heckbert, Surface simplification using quadric error metrics PhD thesis, Carnegie Mellon University, CS Department, 1998.

[16] M.Garland Quadric-Based Polygonal Surface Simplification In *SIGGRAPH 97 Proc.*, pages 209–216,August 1997.

[17] Paul S. Heckbert and Michael Garland Optimal Triangulation and Quadric-Based Surface Simplification

[18] Paul S. Heckbert and Michael Garland Survey of Polygonal Surface Simplification Algorithms

[19] Thomas A.Funkhouse Carlo H.Sequin and Seth J. Teller Management of large amounts of data in interactive building walkthroughs, In 1992 Symposium o Interactive 3D Graphics, pages 11-20, 1992 Special issue of Computer Graphics

[20] Steve Upstill. The Renderman Companion. Addison Wesley, Reading, MA 90

[21] John Rohlf and James Helman. IRIS Performance: A high performance multiprocessing toolkit for real-time 3D graphics. In SIGGRAPH'90 Proc, pages 381-394 July 1994.

[22] Michael A.Cosman and Rovert A.Schumacker. System strategies to optimize CIG image content. In Proceedings of Image II conference, pages 463-480. Image Society, Tempe, AZ, June 1981

[23] Peter Lindstrom, David Koller, William Ribarsky, Larry F.Hodges, Nick Faust and Gregory A,Turner. Real- time, continuous level of detail rendering of height fields, In SIGGRAPH'96, pages 109-118, August 1996.

[24] Mark Duchaineau, Murray Wolinsky, David E.Sigeti, Mark C.Miller, Charles Aldrich and Mark B.Mineev-Weinstein. ROAMing terrain: Real- time optimally adapting meshes. In IEEE Visualization 97 Conference Proceedings, pages 81-88, October 1997.

[25] Leila De Floriani, Enrico Poppo and Paola Magillo. A formal approach to multi-resolution by persurface modeling. In W.StraBer, R.Klein nad R.Rau, editors, Geometric Modeling: Theory and Practice, SpringerVerlag, 1997.

[26] Leila DeFloriani, paola Magillo and Enrico Puppo. Efficient implementation of multi-triangulations . In IEEE Visualization 98 Conference Proceedings, pages 43-50, Oct 1997

[27] W.J.Schroeder, J.A.Zarge, and W.E.Lorenson. Decimation of triangle meshes. Computer Graphics, 1992, 26(2): 65-70.

[28] M.Krus, P.Bourdot, F.Guisnel, and G.Thibault. Levels of detail & Polygonal Simplification.

[29] M.Eck, T.DeRose, T.Duchamp, H.Hoppe, M.Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitray meshes. In Proceedings of SIGGRAPH'95, 1995: 173-182.

[30] A.D.Kalvin, R.H.Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. IEEE Computer Graphics and Applications, 1996: 64-77.

[31] T.A.Funkhouser and C.H.Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In Proceedings of SIGGRAPH'93, 1993: 247-254.

[32] P.Lindstrom, D. Koller, W.Ribarsky, L.F.Hodges, N.Faust, and G.A.Turner. Real-Time Continuous Level of Detail Rendering of Height Fields. In Proceedings of SIGGRAPH'96, 1996: 109-118.

[33] J.Xia, and A.Varshney, Dynamic View-dependent Simplification for Polygonal Models, IEEE, Visualization'96 Proceedings:327-334

[34] D.Luebke, and C.Erikson, View-Dependent Simplification of Arbitrary Polygonal Environments, Computer Graphics, SIGGRAPH'97 Proceedings

[35] 石教英 虚拟现实基础及实用算法 科学出版社 2002

[36] 潘志庚, 马小虎, 石教英. 虚拟环境中多细节层次模型自动生成算法. 软件学报, 1996, 7(9): 526-531.

[37] 潘志庚,马小虎,石教英,虚拟现实中多细节层次模型自动生成技术综述,中国图像图形学报,1998,3(9):754-759).

[38] 周晓云,刘慎权. 基于特征角准则的多面体模型简化方法. 计算机学报(增刊), 1996, 19(9): 217-223.

[39] M.Eck, T.DeRose, T.Duchamp, H.Hoppe, M.Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitray meshes. In Proceedings of SIGGRAPH'95, 1995: 173-182.

[40] Steve Upstill. The Renderman Companion. Addison Wesley, Reading, MA, 1990.

[41] John Rohlf and James Helman. IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In SIGGRAPH '94 Proc., pages 381{394, July 1994.

[42] Michael A. Cosman and Robert A. Schumacker. System strategies to optimize CIG image content. In Proceedings of the Image II Conference, pages 463{480. Image Society, Tempe, AZ, June 1981.

[43] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: Realtime optimally adapting meshes. In IEEE Visualization 97 Conference Proceedings, pages 81{88, October 1997.

[44] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height _elds. In SIGGRAPH '96, pages 109-118, August 1996.

[45] Leila De Floriani, Paola Magillo, and Enrico Puppo. E_cient implementation of Multi-triangulations. In IEEE Visualization 98 Conference Proceedings, pages 43{50,517, Oct 1998.

[46] Leila De Floriani, Enrico Poppo, and Paola Magillo. A formal approach to multiresolution hypersurface modeling. In W. Stra_er, R. Klein, and R. Rau, editors, Geometric Modeling: Theory and Practice. Springer- Verlag, 1997.

[47] C.Wiley, S.Szygenda, D.Fussell, and F.Hudson. Multi-resolution BSP Trees Applied to Terrain, Transparency, and General Objects.

[48] P.Cignoni, C.Montani, C.Rocchini, R.Scopogno, Resolution Modeling. Technical Report C97-02, *CNUCE-C.N.R., Pisa, Italy, 1997*

[49] 秦绪佳 刘新国 鲍虎军 彭群生 网格的渐进几何压缩 软件学报 2002 Vol.13, No. 9

[50] 石教英. 虚拟环境(VE)技术及其应用. 全国第一届虚拟环境研讨会论文集, 杭州,1994.

[51] 陶志良, 潘志庚, 石教英, 基于能量评估的网格简化算法及其应用, 软件学报, 1997, 8(12):881-888