高等学校计算机专业规划教材

现代计算机图形学基础

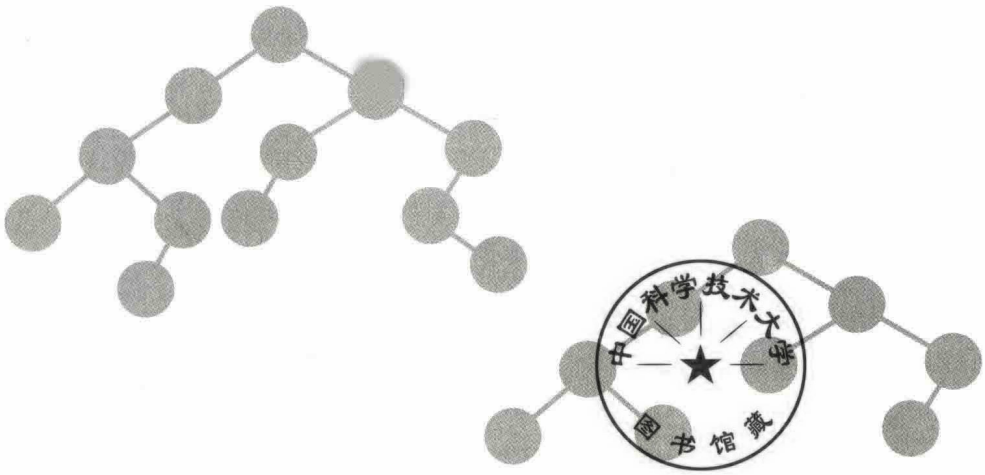
黄华 张磊 编著

清华大学出版社

高等学校计算机专业规划教材

现代计算机图形学基础

黄华 张磊 编著



清华大学出版社
北京

内 容 简 介

计算机图形学是研究真实或虚拟物体在计算机中的图形表示及交互的一门学科，在计算机辅助设计与制造、计算机动画、计算机游戏、计算机仿真、虚拟现实、科学计算可视化等领域有着广泛的应用。随着数字媒体、虚拟/增强/混合现实、大数据、人工智能、可视分析等技术的发展，计算机图形学的理论和方法不断地充实与更新，已经广泛应用到大量的科学和工程领域，成为现代计算机应用中不可缺少的重要分支。

本书涵盖了计算机图形学的众多基础内容，不仅介绍了几何建模、真实感绘制、计算机动画等传统图形学的内容，还着重阐述了数字几何处理、非真实感绘制、基于图形的影像处理、计算摄像、GPU 图形计算等图形学的新进展。其中，不少章节的内容都取自近二十年内计算机图形学领域的高水平学术论文和产业技术。此外，本书还给出了很多重要概念和方法的实例，并提供了一些代表性算法的关键实现步骤的代码。

本书不仅可以作为高等院校计算机相关专业的“计算机图形学”课程教材，也可以作为计算机图形学科研工作者和产业界技术人员的参考用书。

图书在版编目(CIP)数据

现代计算机图形学基础/黄华，张磊编著. —北京：清华大学出版社，2020.5
高等学校计算机专业规划教材
ISBN 978-7-302-55271-0

Ⅰ. ①现… Ⅱ. ①黄… ②张… Ⅲ. ①计算机图形学 Ⅳ. ①TP391.411

中国版本图书馆 CIP 数据核字(2020)第 049745 号

责任编辑：龙启铭
封面设计：何凤霞
责任校对：焦丽丽
责任印制：丛怀宇

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-83470236
印 装 者：三河市龙大印装有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：15.75　　**字　　数**：360 千字
版　　次：2020 年 6 月第 1 版　　**印　　次**：2020 年 6 月第1次印刷
定　　价：59.00 元

产品编号：083146-01

序

计算机图形学起源于图灵奖获得者、美国科学院和工程院院士 Ivan Surtherland 在 20 世纪 60 年代发明的交互式绘图系统——“画板”。在过去半个世纪的时间里，计算机图形学的研究与发展可谓“一日千里”，并且在 CAD/CAM/CAE、影视制作、计算机游戏、计算机仿真等领域有着广泛的应用。事实上，计算机图形学已经成为相对成熟的学科，理论和方法精彩纷呈，是计算机科学中最活跃的分支，也是计算机科学和工程应用之间的桥梁。

20 世纪 80—90 年代，是国际上计算机图形学蓬勃发展的黄金时期，而国内则处于刚起步阶段。那时，有关计算机图形学的中文教材和参考书甚少，使得图形学的入门较为困难。为了吸引学生和科技工作者投入到计算机图形学，推动国内计算机图形学的发展，老一辈的学者高瞻远瞩，开始翻译或者编著计算机图形学教材，例如北京航空航天大学的唐荣锡教授，清华大学的唐泽圣教授、孙家广教授，浙江大学的梁友栋教授、石教英教授、彭群生教授、汪国昭教授等前辈。今天看来，这些经典的图形学教材，对推动当时国内计算机图形学的普及和发展起到了很好的作用，培养了一大批的图形工作者。

近二十年来，计算机科学和技术发生了巨大的变化，以人工智能、云计算、大数据、虚拟现实与增强现实等为代表的新技术也带动了计算机图形学的进一步发展，涌现出了更加激动人心的图形学浪潮。然而，目前的图形学教材略显滞后，尚未及时整理和吸收这些新发展。这就迫切需要一本能够反映图形学崭新内容的教材，以使初学者或者具有图形学基础知识的人能够及时掌握这些新发展。

本书的出现，恰好能够弥补上述的不足。本书对以往图形学教材的内容做了大幅的扩展，尤其是涵盖了许多计算机图形学的新发展。这是本书的一大特色，对新形势下计算机图形学学科的传承与发扬具有重要的意义。具体到章节安排上，本书的第 2、3、5 章是经典图形学的建模和绘制内容，而其余章节则详细介绍了近二十年计算机图形学在方法和应用方面取得的新进展，包括数字几何处理、非真实感绘制、基于图形的影像处理、计算摄像等内容。这为初学者快速掌握最新的计算机图形学理论、方法和技术，提供了很好的入门教材和参考资料。

除了理论和方法学习，计算机图形学也是一门很注重实践的学科，需要

通过编程来实现各种图形学算法。因此，本书列举了很多的例题，并在书中给出了一些代表性算法关键步骤的伪代码。相应的源代码也提供了网络下载。这对初学者正确理解和快速掌握本书中所介绍的图形学方法和技术大有裨益。同时，每章后的思考题和书后的参考文献也能引发读者继续深入学习。

值得一提的是，本书的作者黄华教授和张磊副教授长期从事计算机图形学的教学和科研工作，在国内外重要学术期刊和会议上发表了大量的论文。他们对经典计算机图形学的内容和现代计算机图形学的新进展都有着深入的研究和体会，在此基础上汇聚成了本书的内容。虽不一而足，但仍可借鉴。因此，相信本书的问世，将能够及时、全面、翔实地介绍计算机图形学的过去和现在的内容，为初学者或者具有一定图形学基础的低年级学生提供一本新的教材和参考书，推进计算机图形学的进一步发展。

胡事民

清华大学

计算机科学与技术系

2020 年 1 月

前言

我从2013年开始接手研究生“计算机图形学”课程的教学工作。甫一接手就寻找合适的教材，以满足研究生培养的要求。说实话，我并没找到一本合适的教材。主要原因是，目前市面上大多数的图形学教材内容比较陈旧，基本都是传统图形学的内容，而最近的一二十年，计算机图形学突飞猛进。传统图形学的建模、绘制的很多内容都已经被集成到GPU中，普通人员接触不到，而图形学近二十年的进展在目前教材中体现过少。作为研究生，必须了解较新的学科发展状况。从那时起就萌发了编写“计算机图形学”课程讲义的想法，让研究生通过这门课程既能掌握经典图形学的内容，也能了解现代图形学的新进展。

还是在2013年5月，清华大学胡事民教授将他最近几年给研究生讲授“计算机图形学”课程的课件提供给了我，同时，我也收集了其他兄弟高校的课件。基于这些课件我完成了自己的课件。此后每年授课过程中，都会对课件进行修改和完善。

2016年秋，我获批了学校图形学与虚拟现实研究生教学团队建设项目，编写“计算机图形学”课程的讲义是该项目的目标之一。于是在2016年冬天，我们邀请浙江大学鲍虎军教授、清华大学胡事民教授和雍俊海教授、山东大学张彩明教授等国内计算机图形学领域的资深专家，对我们的课程讲义大纲进行了评审。与会专家充分肯定了出版一本适合研究生教学用的图形学教材非常必要，并对课程的内容提出了大量的建设性意见。这让我们深受鼓舞。

此后，我们就开始根据课程讲义大纲编写教材，虽然已经有了课件，但编写起来还是非常耗时耗力的，毕竟落在纸面上，需要尽可能地准确和严谨。这样，经过两年的辛勤努力，到了2018年冬天，终于拿出了一个初稿，并将书名定为《现代计算机图形学基础》。2018年12月，我们又请清华大学胡事民教授和浙江大学鲍虎军教授对初稿进行了审阅，两位教授再次提出了一些宝贵意见，特别是鲍虎军教授对教材的章节安排给出了重要建议。

感谢清华大学出版社龙启铭先生。因为此前有过合作，在2018年冬天，和龙启铭先生聊天时谈起我们正在编写“计算机图形学”课程的教材。不曾想龙启铭先生非常积极，提出看看书稿，书稿看完后，就提议由清华大学出版社出版，并提出了部分章节内容扩充、提供伪代码以及插图制作等方面的建议。

我们在此基础上进一步对书稿进行了修改完善，终在2019年秋天完成了书稿第一版定稿。

本书中，除了传统计算机图形的几何建模、真实感绘制、计算机动画等内容外，还包括了数字几何处理、非真实感绘制、基于图形的影像处理、计算摄像、GPU图形计算等，不少内容都取自近二十年图形学方向的高水平论文，可充分反映计算机图形学的新进展。同时，在教材编写过程中，考虑传统图形学的建模、绘制等内容都已经被集成到GPU中了，我们较大幅度压缩了这部分的内容。

本书既可以用于研究生教材和计算机图形学相关方向的科研工作者查阅，也可以选择部分内容作为本科生教材。本书配套的源代码、课件等都通过网络提供。

因才疏学浅，且时间、精力有限，本书中如出现错误之处，敬请指出，以便后续完善。

最后，衷心感谢浙江大学鲍虎军教授、清华大学胡事民教授等专家，清华大学出版社龙启铭先生，以及其他对本书的出版给予支持和帮助的所有人。没有你们，就不会有这本教材。

黄　华

2020年1月

目录

第 1 章

引　言

计算机图形学(Computer Graphics,CG),是研究真实或虚拟世界在计算机中的图形表示及人机交互的一门学科。计算机图形学涉及计算机、数学、物理学、心理学、控制科学等学科领域的知识。从 20 世纪 60 年代计算机图形学诞生开始,经过科研工作者和工程技术人员数十年的研究与实践,计算机图形学的理论、方法和技术日益成熟,并广泛应用在工业、军事、医学、建筑、影视、文体、娱乐等行业。近些年来,伴随着物联网、大数据、云计算、边缘计算、人工智能等新兴技术和相关产业的大范围推广,计算机图形学从内涵到外延都发生了巨大的变化。

本章介绍经典计算机图形学的基本概念、计算机图形学的历史发展,尤其是不同阶段计算机图形学所取得的代表性技术成就。进而,通过分析输入、处理和输出方式的差异,介绍计算机图形学与计算机视觉、图像处理等相关学科领域的联系。最后简要介绍计算机图形学在不同行业的应用情况。

1.1　计算机图形学的概念

如图 1.1 所示,传统计算机图形学的内涵包括建模、绘制和交互三个方面。

图 1.1　计算机图形学中的建模、绘制与交互技术

建模,是指在计算机中通过几何、图像、视频等可视媒体形式构造和表示真实或虚拟物体和场景的模型,其目的是形成三维的数据表示形式,也称为三维模型。例如,借助 3d Max 建模软件,可以在计算机中采用点、线、面等基本几何元素,表示一架真实飞机的三维模型。在此基础上,工程师就可以通过键盘、鼠标等输入设备对该三维模型进行平移、

旋转、缩放等三维操作，从各个角度观看飞机的形状。

绘制，是指通过可视媒体形式来呈现计算机中表示的物体和场景模型，其目的是形成人类视觉通道可以直接感知并理解的信号。例如，通过选择合适的绘制算法，可以将真实世界中实际不存在的一些物体在屏幕上呈现，并形成尽可能逼真的画面。

交互，是指通过各种输入和输出设备，让用户以最有效的方式对计算机中的模型进行操作，并能够通过及时的绘制在终端上显示交互结果。例如，采用体感设备 Kinect，可以通过人体姿态的变化控制计算机中的模型，从而达到非接触式的人机交互操作效果。

随着多媒体、虚拟现实、大数据、人工智能等技术的发展，现代计算机图形学的内容也得到扩展，其外延也触及更多的计算机应用领域。例如，计算机动画、科学可视化、影像处理、虚拟现实、增强现实、计算机仿真等领域都与计算机图形学有着密切的联系，使得计算机图形学发展成以建模、绘制、交互、动画、影像处理等为核心内容，同时与其他领域广泛交叉的学科。这里面，交互已经单独发展为一门独立的学科，专门研究系统与用户之间的交互关系，而不再局限于人和计算机之间的交互。因此，交互将不会作为本书的重点内容进行介绍。总之，计算机图形学已经成为现代计算机科学与技术不可缺少的重要分支。

1.2 计算机图形学的发展

“计算机图形学”这个名词，最早是 1960 年由美国波音公司的工程师 William Fetter 提出，用于描述飞机驾驶室模拟设计的过程。1964 年，Fetter 设计了第一个计算机人体模型，用于更好地设计飞机驾驶舱。作为一种计算机应用技术，公认的计算机图形学诞生的标志是 1963 年 Sketchpad 画板系统的问世。该系统是美国 MIT 大学的 Ivan Sutherland 所开发。它实现了一种基于画笔交互的图形界面设计功能（如图 1.2 所示）。Sketchpad 成为有史以来第一个计算机交互式绘图系统，也被认为是人类“曾经编写过的程序中最重要的一份程序”。因此，Sutherland 也于 1988 年获得了计算机科学领域的最高奖项——图灵奖。

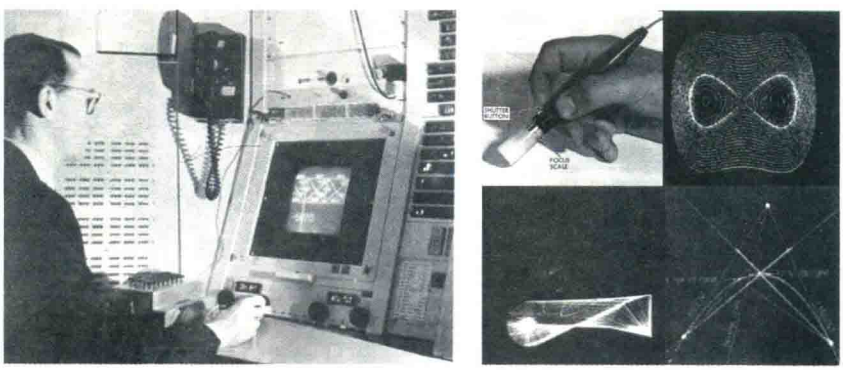


图 1.2 Sketchpad 画板系统（图片来自参考文献[1]）

从人工智能的范畴来讲，Sketchpad 或者说计算机图形学的出现，更重要的意义在于使计算机开始具备了人类右脑的部分功能。这是因为按照诺贝尔生理学和医学奖得主、

美国心理生物学家斯佩里博士的“左右脑分工理论”，人类存在左右脑功能划分，其中，人脑左半球的主要功能是进行逻辑推理和语言表达，也就是与语言、数字以及概念、分析等功能有关；而右半球的主要功能是空间和形象的思维，也就是体现在绘画、直觉、空间感、整体性以及想象和综合等方面的能力。在图形学出现之前，计算机主要是围绕符号处理进行系统设计与使用，而图形学的出现则使计算机具备了图形图像处理的能力，让人和计算机能够更加自然、便利地交流。因此，可以说计算机图形学使得计算机开始实现右脑的部分功能。

从20世纪60年代计算机图形学诞生开始，计算机图形学的建模、绘制及交互技术就不断地往前发展，而且它们之间也是互相促进。图1.3展示了计算机图形学发展历程中的若干代表性技术。从20世纪60年代到现在，建模、绘制和交互技术都发生了巨大的变化，直接或间接地推动了计算机图形学的阶段性发展及其在各个行业中的应用。

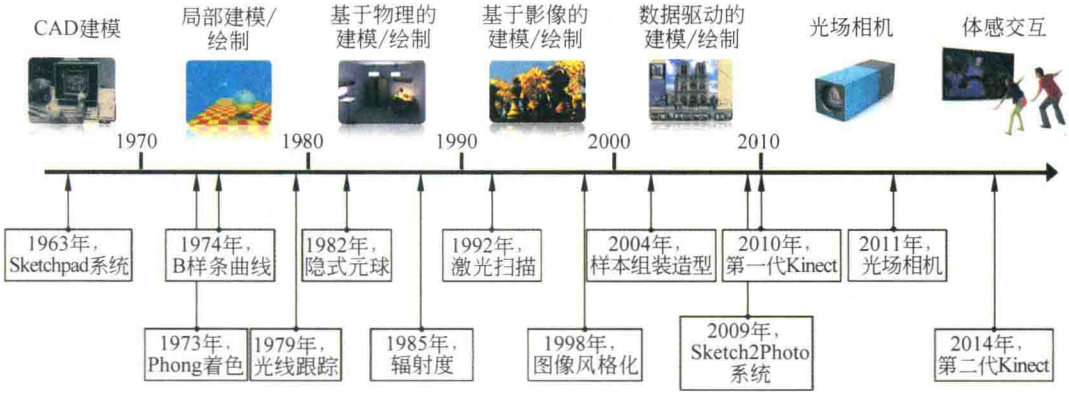


图1.3 计算机图形学的简明发展历程

接下来以时间为序，介绍这些技术在不同阶段的发展情况、取得的成就和带动的相关行业应用，从而更好地了解计算机图形学学科的整个历史以及现状。

1.2.1 20世纪60年代

1. 建模

如何对外形进行精确的数学表示和工业设计，是这个时期建模技术主要解决的问题。在汽车、飞机、轮船等的工业制造过程中，通常需要非常准确且灵活的造型工具，其目的是能构建满足指定几何性质的外形，并且方便工程技术人员随时进行修改。1962年，法国雷诺公司的工程师Pierre Bézier提出了Bézier自由曲线技术。Bézier曲线提供了一种基于控制顶点的曲线参数化表示形式，本质上是Bernstein多项式组合形成的参数化表达式。在自由曲面建模方面，MIT的Steven Anson Coons提出了Coons patch自由曲面造型技术。Coons曲面本质上是一种基于控制顶点的双变量曲面参数化表示形式，能够在曲面角点和边界处进行灵活地设计，以满足工业产品设计时的几何性质。这两种自由曲线曲面是计算机辅助设计(Computer Aided Design，CAD)和计算机辅助制造(Computer Aided Manufacturing，CAM)中的典型建模技术，至今仍被广泛使用。

为了纪念 Bézier 及其创造性的贡献，美国工业与应用数学学会(SIAM)自 2007 年起设立了 Pierre Bézier 奖，用于奖励在实体、几何和物理建模及应用领域做出突出贡献的人。而从 1983 年开始，美国计算机图形学协会(SIGGRAPH)设立了 Steven Anson Coons 奖，以纪念 Coons 及其贡献。该奖项每奇数年颁发一次，用于表彰在计算机图形学和人机交互领域具有持续贡献的人，被认为是计算机图形学领域的成就奖。

2. 绘制

这个时期的绘制技术主要面向 CAD 和 CAM 系统建模获得的几何模型的快速绘制和屏幕显示。例如，线段、三角形、圆、椭圆等基本几何形状的快速绘制，以及由这些基本几何元素进一步组合而构成的更复杂的几何形状的显示。由于计算机屏幕是以像素为单元进行图形显示，在绘制时需要解决连续几何形状的离散表示，也就是基本的几何元素到像素的转化。例如，如何用离散的像素表示连续的一条线段而不会产生太大失真，以及如何在有限分辨率的屏幕更好地显示几何形状。1962 年，美国 IBM 公司的 Jack Bresenham 设计了针对有限长度线段的绘制算法。后来在美国 Harvard 大学工作的 Ivan Sutherland 和他的学生 Danny Cohen，提出了针对计算机矩形屏幕显示的线段裁剪方法，能够对位于线段上的像素进行准确而高效的绘制。这两个工作解决了计算机对线段这一最基本几何形状进行快速而准确绘制的问题。受限于当时计算机的运行速度，这些特定绘制技术的提出，有力地提高了计算机图形显示的能力，为后续计算机图形学的发展奠定了坚实的基础。

3. 交互

早期的计算机是通过纸带作为输入和输出的媒介。而在 20 世纪 60 年代的时候，键盘(打字机)已经作为这个时期主要的终端输入。然而，1963 年 Ivan Sutherland 发明的画板系统 Sketchpad，创造性地使用了光笔进行图形输入，同时也能在屏幕上直接输出图像。用户在使用该系统时，可以直接通过 X-Y 绘图仪显示光笔位置，从而使得这种笔式人机交互相比于键盘输入更为便捷，为人和计算机之间提供了一种全新的交互方式，也为后续人机交互技术的发展开辟了一条新路。

1.2.2 20 世纪 70 年代

1. 建模

针对 Bézier 曲线缺乏形状局部控制的缺陷，1974 年，美国 Utah 大学的 William Gordon 和 Richard Riesenfeld 将 B 样条参数曲线引入几何造型。B 样条曲线本质是分段多项式曲线，具有局部支撑性、仿射不变性、凸包性等许多良好的性质，也可以构造 B 样条曲面。随后，B 样条曲线/曲面逐渐成为新的建模工具。但是，B 样条曲面在计算机内部处理时，需要经历从连续到离散的过程。因此，一种基于离散形式的细分曲面建模技术被提出。例如 1978 年，美国纽约理工大学的 Edwin Catmull 和 Jim Clark 基于双三次均匀 B 样条曲面递归计算过程，提出了 Catmull-Clark 细分曲面；英国 Brunel 大学的 Daniel Doo 和 Malcolm Sabin 基于双二次均匀 B 样条曲线递归模型，提出了 Doo-Sabin 细分曲面。这类细分曲面通过一系列中间状态的网格来逼近目标形状，克服了自由曲线曲面的离散转化问题。细分曲面被广泛用于计算机动画的角色外形设计，而 B 样条曲线/曲面

则逐渐成为工业 CAD 中外形设计的标准工具。

2. 绘制

由于没有 GPU 等专用图形计算架构，如何用 CPU 有限的计算能力尽可能真实地绘制计算机中的三维模型，是这个时期图形绘制面临的重要问题。20 世纪 70 年代初期，Utah 大学的 Henri Gouraud、Bui Tuong Phong 等人先后通过对三维模型顶点颜色、法向等的插值，建立了高效的局部光照模型，能够对简单场景进行高效地真实感绘制。此外，Utah 大学的 Edwin Catmull 等人提出了纹理映射技术，能够借助真实的图像对简单的几何场景进行复杂的绘制显示。为了进一步提高真实感，作为整体光照模型的光线跟踪技术被提出。1979 年，美国 Bell 实验室的 Turner Whitted 改进了之前的光线投射算法，采用从视点位置投射光线的跟踪算法进行真实感的图形绘制，能够实现反射、折射等光照效果，使得三维模型的绘制效果更加接近于真实世界中光线和物体的物理作用效果。伴随着这些真实感绘制技术的出现，计算机图形学在传统 CAD 和 CAM 等行业外，开始跨入影视和游戏行业的制作过程。

3. 交互

鼠标的出现为这一时期的人和计算机交互提供了新的方式。鼠标最早可以追溯到 1964 年美国加州大学伯克利分校的 Douglas Engelbart 博士发明的装有滚轮的木头盒子。1973 年，美国施乐公司 Palo Alto 研究中心引入 Xerox Alto 计算机作为工业设计和制造的机器，并使用鼠标作为输入工具之一，这也是第一台使用鼠标的计算机。鼠标的使用，有效地取代了传统只通过键盘输入繁琐指令的交互方式。1977 年，美国 Apple 公司推出了 Apple Ⅱ计算机，造就了集成键盘式计算机的巨大成功。同时，该型号的计算机也配备了鼠标作为键盘之外的辅助输入设备。这些面向大众的商业计算机的成功，也有力地推动了人机交互技术的普及。

1.2.3 20 世纪 80 年代

1. 建模

为了解决 Bézier 曲线曲面、B 样条曲线曲面等自由曲线曲面定义域拓扑受限的问题，隐式曲线曲面建模技术被提出，并成为这个时期重要的建模工具之一。1982 年，美国 Caltech 大学喷气动力实验室的 Jim Blinn 提出了基于元球的隐式建模方法，通过球基的代数组合生成光滑的曲面。这种方法最大的优势在于可以表示任意拓扑的几何形状，而不受限于自由曲面定义时所使用的矩形或三角形参数域。然而，隐式曲线曲面毕竟是一种连续函数表示，需要离散化为基本的几何单元，才能被计算机处理。1987 年，美国 GE 公司的 William Lorensen 和 Harvey Cline 提出了著名的 Marching Cube 方法，可以使用离散几何形状有效地逼近连续的隐式曲面，将其转化为满足零阶连续性的离散曲面形式。这样就实现了连续曲面到离散曲面的转化，为隐式建模及其应用提供了有效的工具。

2. 绘制

整体光照模型仍是这一时期图形学真实感绘制技术的核心内容，但人们更关注于如何在满足视觉真实感的同时，使得绘制的效果更加符合真实世界的物理规律。1985 年，美国 Cornell 大学的 Michael Cohen 和 Donald Greenberg 以及日本广岛大学的

Tomoyuki Nishita 提出了辐射度绘制方法。这种方法基于物理学中的热辐射理论，通过模拟两个表面之间光能的传输，建立光线在一个场景里多次反射的模型。这样可以生成更加柔和且自然的阴影和反射效果，进一步提高了绘制场景的真实感。

3. 交互

如何实现更加符合人体功效的输入设备成为这个时期交互技术关注的重要问题。1984 年，IBM 公司发布了第一台配备模型 M 键盘的计算机。在 20 世纪 80 年代中期，美国宇航局艾姆斯研究中心开发了第一个装备在虚拟现实环境中的数据手套，通过感受手指弯曲的程度来控制人和系统的交互。1987 年，美国 Microsoft 公司发布了“视窗 2”桌面系统，允许重叠的窗口显示，使得丰富多样的图形界面进入计算机操作系统，大大方便了人机交互，也逐渐成为使用最广泛的桌面操作系统。

1.2.4 20 世纪 90 年代

1. 建模

随着以自由曲线曲面为工具的几何建模方式逐渐成熟，国际标准化组织（ISO）在 1991 年将非均匀有理 B 样条曲线曲面定义为 CAD 系统中工业设计的标准，它至今仍是工业界最流行的几何建模工具之一。而随着光学测量技术的发展，尤其是以激光测距为代表的三维扫描技术的推广，直接扫描真实世界的物体并转化成便于计算机表示及处理的三维模型，成为这个时期另外一种重要的建模手段。1992 年，美国 Stanford 大学的 Marc Levoy 和他的学生研究了用激光扫描仪进行数字化形状建模的方法。1997 年，由该大学牵头的“数字米开朗基罗”工程开始实施，其目的是通过三维扫描将文艺复兴时期的雕塑进行数字化表示，在计算机内将各种尺寸的雕塑进行形状数字化，并在计算机上进行存储。这样就能够实现文物保护的信息化，为人类文明的传承和发展提供新的服务模式。

2. 绘制

随着数码摄像设备的普及，直接获取真实的照片变得非常容易。如何利用真实照片直接进行场景或模型的绘制，成为新兴的技术。1995 年，美国 UIUC 大学的 Leonard McMillan 和 Gary Bishop 开发了一个基于图像的渲染系统——基于全光场的绘制。1998 年，美国纽约大学 Aaron Hertzmann 提出了图像风格化的方法，将输入的真实图像转化成油画、水彩画、素描画等具有一定艺术风格的图像，开创了计算机图形学领域非真实感绘制的研究方向。与真实感绘制不同，非真实感绘制侧重于采用具有一定艺术效果的方式对二维图像或三维模型进行绘制，从而方便了普通用户制作艺术图片。

3. 交互

如何突破传统依赖物理媒介进行交互的方式，成为这一时期关注的新问题。1992 年，UIUC 大学创建了一个自动的虚拟环境工作室，这个工作室是一个方形房间，通过广角投影呈现一种虚拟的环境。1993 年，IBM 公司发布了第一部没有物理按键、完全靠触摸屏操作的手机——IBM Simon，开创了移动终端触摸交互方式。

1.2.5 21世纪

1. 建模

随着几何模型数据规模的扩大，采用数据驱动的方式进行建模成为新的方式。2004年，美国Princeton大学的Thomas Funkhouser等人提出了一种基于样本的几何建模方法，从已有的几何模型生成新的几何模型。2012年，Stanford大学的研究人员提出了基于概率的形状合成方法，从有限的形状集合出发，按照部件组合的概率生成新的模型。另一方面，移动平台上的激光测距、摄影测量等技术的广泛使用，也为大范围的场景建模提供了有效手段。2005年，美国Google公司发布了谷歌地球，支持建筑和山脉的三维模型浏览。同年，Microsoft公司也发布了类似的虚拟地球平台。这两个平台上的地形、建筑物等三维模型大多采用大范围的三维扫描生成，支持用户从多个不同视角浏览这些三维模型。

2. 绘制

随着互联网上海量影像数据的聚合，利用网络图像进行场景绘制发展为新的趋势。2006年，美国Washington大学的Noah Snavely等人开发了一套基于三维模型的图像浏览系统，使得用户足不出户便可观赏互联网上各地的风景名胜影像。2007年，美国CMU大学的James Hays采用海量互联网图像作为素材，解决图像补洞问题。2011年，清华大学计算机系开发了一套基于互联网图像的图像生成系统Sketch2Photo，利用草图作为输入，对互联网图像中的物体和背景进行合成，能够实现简单笔画交互下真实图像的生成。在此基础上，2013年清华大学又推出了从输入的草图直接生成三维场景的系统Sketch2Scene，能够自动地将语义相关的三维模型进行组合和绘制，从而帮助用户更便捷地获取复杂三维场景画面。

3. 交互

多点触控和非接触式交互为用户提供了新的体验。2007年，Apple公司发布第一代iPhone手机，支持图片在手机屏幕上的多点触控，允许更灵活的平面操控。2010年，Microsoft公司发布了Kinect体感设备，通过红外激光进行即时动态捕捉，实现了不与任意设备进行接触情况下的人机交互。2013年，美国Leap Motion公司发布了支持手部姿态作为输入的体感设备，使得通过手部运动进行非接触式交互变得更加便捷。2017年，伴随着Apple公司的iPhone X手机的问世，深度摄像头也逐渐变成智能手机的标准配件，为借助手势、人脸等生物特征进行交互提供了新途径。

1.3 计算机图形学与其他学科的关系

在计算机学科范围内，计算机图形学和计算机视觉、图像处理是联系最紧密的三个学科方向，同属计算机应用领域。经典的计算机图形学以建模、绘制和交互等内容作为内涵。在不同的应用中，通常是在三维建模后，将模型绘制成屏幕图像进行呈现，同时可以借助各种输入设备进行人机交互，对三维模型进行操作。简单地说，计算机图形学的输入是三维模型，输出是二维图像。计算机视觉则是指用计算机模拟人类的视觉功能，也就是

通过计算机的处理实现对客观三维世界中场景的主观感知、加工和理解，从而让计算机能够实现人类视觉的一些功能。计算机视觉输入的是二维图像，输出的则是对客观三维世界场景的理解。因此，从工程的角度分析，计算机视觉可以看作是计算机图形学的反问题。图像处理则是对图像本身进行各种计算和处理，例如对图像本身的颜色、光照、几何等信息进行重组和再生成。因此，图像处理的输入是图像，输出仍是图像。

如图 1.4 所示，从人类视觉系统组成及功能来看，人对周围环境形成视觉感知的过程，可以看作将客观世界物理系统中的物体，经过人眼光学系统在视网膜成像，然后通过神经组织送入大脑相应功能区进行处理。那么，计算机图形学对应于从物理系统到光学系统成像的过程，将三维场景映射成二维图像，形成可感知的影像信号。计算机视觉则是从视网膜成像后信号输入到大脑感知系统并进行处理的过程，也就是对二维图像本身内容的理解，最终形成相应的知识，实现计算机从感知到认知的演化。因此，计算机图形学和计算机视觉以及图像处理，是紧密相关的学科，彼此之间既互为补充，又存在很多交叉内容。

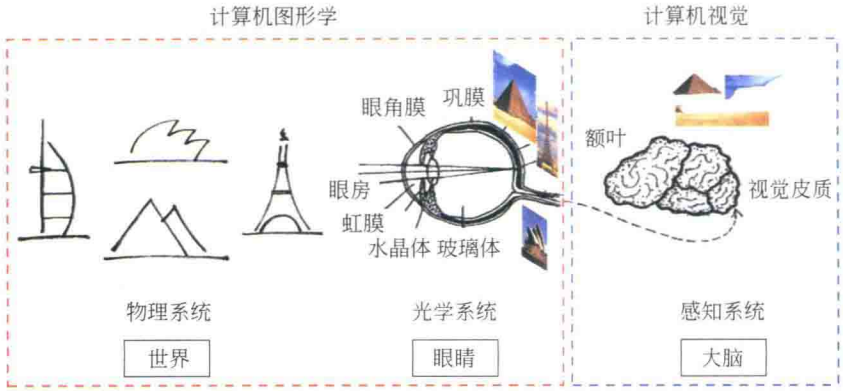


图 1.4 人类视觉感知系统

随着计算机新技术的不断涌现，计算机图形学也面临着许多新的问题和挑战。2009 年 Steven Anson Coons 奖获得者、美国 Pixar 公司的 Rob Cook 在当年 SIGGRAPH 大会上的演讲中高屋建瓴地总结了下个三十年，计算机图形学需要去关注和解决的十个重要的科学和技术问题。这些问题包括自然的三维交互、直观的三维编辑、富有感情的机器人技术、绘画辅助技术、数学辅助技术、海量数据的可视化、虚拟导游技术、高沉浸感技术、增强现实技术、混合现实技术等。

具体来讲，自然的三维交互，是指要从传统鼠标、单点触摸、多点触摸的二维接触式交互方式转变为基于手势、姿态等的三维非接触式交互方式，突破现有的人和计算机之间的交互模态。直观的三维编辑，是要通过画笔手绘等常见且直观的艺术创作手法，对计算机中的三维模型进行建模或修改，达到对三维模型灵活自如的编辑。富有感情的机器人技术，是指要用技术赋予计算机或机器人以人类社会活动时的情感，使之在工作时能够表达、识别和理解喜怒哀乐，并具有模仿、延伸和扩展人的情感的能力，制造拥有类人情感的机器人。绘画辅助技术，是指能够通过直线、曲线、笔画、素描等不同形式的艺术绘画手

段，让计算机更加智能地辅助人工参与的绘画创作。数学辅助技术，是指希望通过图形方式辅助论文编辑、数学软件使用、数学推理等过程，让繁杂的数学工作变得更加轻松。海量数据的可视化，是指要借助面向海量数据的可视化技术，从根本上改变人们表示、分析和理解海量、复杂数据的方式，实现对海量数据的视觉感知。虚拟导游技术，则是指利用计算机提供的智能导游服务取代人力服务，将各种各样的虚拟角色融入吃、穿、住、用、行的虚拟现实场景中，从而大大提高服务质量。高沉浸感技术，是指希望提供一种能使用户在专注当前虚拟和仿真的情境时感到愉悦和满足的技术，以达到虚拟环境下沉浸式的体验和交互。增强现实和混合现实技术，是指要形成与真实世界无缝融合、高品质的完全沉浸式的数字世界，并通过更为人性化的交互构建现实世界、虚拟世界和用户之间的信息回路，实现真正意义上的虚实融合。可以预见，这些重要技术问题的解决也必然会推动计算机图形学的新发展。

1.4 计算机图形学的应用

计算机图形学是伴随着工业设计、电影制作、计算机游戏、虚拟现实等产业而兴起和繁荣的。如图 1.5 所示，计算机图形学在这些产业的发展过程中也不断融合其他科学技术的先进成果，在相关领域取得了广泛的应用，既有力地推动了这些产业的蓬勃发展，也为计算机图形学自身的发展提供了源动力。

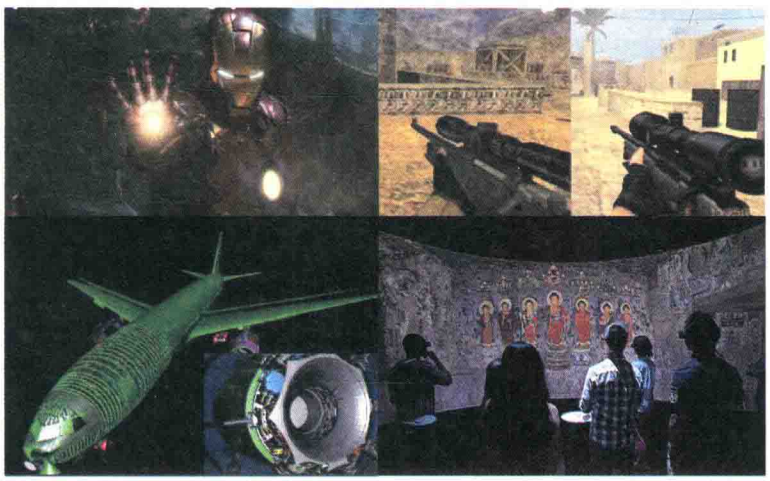


图 1.5 计算机图形学在一些行业中的应用

1. 工业设计和产品制造

工业设计和产品制造过程中广泛使用了 CAD/CAM 等软件进行外形设计等任务，其实质是几何建模。例如，波音飞机公司使用法国达索公司的 CAD 软件 CATIA，完成了整个波音 777 机型的电子装配，创造了世界航空业的一个奇迹。在工业设计时，采用这些软件可以直接在计算机上进行画图、参数修改和仿真，对不同设计方案进行自动分析和比较，并且可以将草图迅速变为电子生产的工作图，方便设计人员快速做出判断和修正，提

高设计的效率，降低工程成本。此外，在逆向工程、三维打印、医疗诊断中也经常用到几何建模技术。

2. 影视媒体

计算机图形学已广泛应用于影视媒体制作中的特效镜头、电影动画等生产过程中。20 世纪 90 年代曾经是计算机动画应用最辉煌的十年，也带动了计算机图形学的蓬勃发展。Pixar 公司(2006 年被美国 Disney 公司收购)每年都要出一部制作精美的卡通动画片，例如《玩具总动员》是世界上第一部完全使用计算机动画技术制作的长篇动画电影，获得了多项奥斯卡奖。后续动画影片，如《海底总动员》《超人总动员》等，也均创造了可观的票房。好莱坞的大片屡屡大量运用计算机生成的各种各样、精彩绝伦的特技效果，例如《星球大战》《加勒比海盗》《变形金刚》等电影，都有美国 Industrial Light and Magic 公司(2012 年被 Disney 公司收购)参与特效制作。

3. 游戏娱乐

现代的三维计算机游戏和手机游戏更是离不开计算机图形学。游戏中的三维角色大多借助于图形建模和绘制技术进行制作，能够呈现不同于现实世界的角色造型和场景画面。逼真的外形和贴近真实的视觉效果，能够让游戏玩家有身临其境的感觉，大大增加了游戏娱乐的体验效果。代表性的计算机游戏 DOOM 的三维引擎，则是大量采用了计算机图形学中的金字塔纹理映射、阴影体等技术来实现多重光影效果，为用户提供了极具真实感的画面，使其成为划时代的一款游戏作品。

4. 虚拟仿真和增强现实

在虚拟仿真和增强现实的诸多应用中，虚拟物体/场景也往往需要借助计算机图形学的建模和绘制技术来实现。例如美国宇航局利用飞行模拟器来模拟飞行环境，使飞行员在室内即可体验起飞、着陆等环境，进行飞行训练。美国 UIUC 大学让医学专业学生在仿真环境下进行临床学习，或者在仿真环境中进行消防演练。美国谷歌公司推出的 VR 版谷歌地球，可以让用户借助虚拟现实眼镜“浸入式”地观看各地景观。北京理工大学光电学院的研究人员对完整的圆明园建筑进行三维重建和仿真，让游客通过望远镜就可以浏览到废墟之上的建筑原貌。

5. 文化和艺术

利用计算机图形学中的几何建模方法重建文物古迹的三维模型，可以借助计算机对文物进行数字存储、分析和观赏，为文化遗产保护提供一种现代化手段。例如，美国 Stanford 大学开展的“数字米开朗基罗”工程，实现了意大利文艺复兴时期雕塑的三维建模和数字浏览。2019 年 4 月 15 日，法国巴黎圣母院突发火灾，造成了令人痛心的建筑损失。事实上，早在 2015 年，美国瓦萨学院的艺术史专家就利用三维激光扫描，保存了巴黎圣母院的三维模型，误差只有约 5mm。这为巴黎圣母院的灾后修复提供了宝贵资料。此外，非真实感绘制技术实现了通过计算机进行智能的绘画创作，甚至可以对手机拍摄的人物、风景图片进行风格化绘制，直接生成艺术作品。2018 年 10 月，纽约佳士得拍卖行对一幅完全由计算机创作的油画进行拍卖，最终以 43 万美元成交。这种艺术风格化绘制技术通过计算机就可以生成艺术作品，能够使艺术创作大众化，让更多的普通人可以随时随地接触到艺术作品，感受到艺术熏陶。

6. 科学探索

将数据和信息通过图形绘制的方式进行可视化展示，能够提升人们对被模拟对象变化过程和规律的认识。因此，计算机图形学中的绘制及可视化技术也被认为是“科学技术之眼”，已经成为科学发现和工程设计的重要工具。例如，通过遥感高光谱数据的真彩色可视化，可以更好地了解地形、地貌及地质条件，为资源勘探、环境监测等提供有效手段。此外，通过对海量、多源、异构数据的可视化展示，可以更好地掌握数据分布规律，挖掘有价值的信息。例如，2019 年 4 月 10 日天文学家首次公布了黑洞的照片，这是花了两年时间、对 5PB(1PB=2^{50}字节)的数据进行分析和可视化而获得的世界上第一张黑洞正面照，将有助于科学家进一步验证爱因斯坦广义相对论等物理学基础理论。

1.5 小 结

本章介绍了计算机图形学的基础知识，包括计算机图形学的基本概念、发展历程、学科特点、应用情况等内容。特别地，围绕传统计算机图形学的三个方面：建模、绘制和交互，以时间为序，回顾了各个历史阶段技术发展和取得成就。在此基础上，引入了现代计算机图形学的发展新阶段、面临的新的问题和挑战，并且着重阐述了计算机图形学在众多行业中的广泛应用。

思考题

1.1 计算机图形学诞生的标志是什么？这对计算机发展的意义是什么？

1.2 建模、绘制和交互技术的发展各自经历了哪些阶段？每个阶段有哪些代表性的技术？

1.3 计算机图形学、计算机视觉和图像处理的联系和区别是什么？

1.4 进入 21 世纪后，计算机图形学的建模、绘制和交互技术有哪些新特点？

1.5 举例说明计算机图形学的若干典型应用。

第 2 章

基 础 知 识

图形是由点、线、面等基本几何元素组合而成的，这些元素统称为图元。在计算机图形学中，图元往往是通过手工建模、设备测量等方式来获得，以此表示物体形状方面的几何信息。然后，这些图元组合在一起形成物体形状的几何模型。复杂的场景又可以通过多个物体模型组合而成。例如，通过测量门、窗、墙、楼梯等的尺寸，可以构建一幢大楼的三维模型。图像则是以像素为基本构成单元，主要通过光学成像等方式获取物体的色彩信息。简单地讲，二维图形在进行放大后不会产生失真，而二维图像在放大后会产生锯齿状的失真。然而，图形在显示器等终端进行呈现时，仍然需要转化为图像，通过 R(红)、G(绿)、B(蓝)三个通道的色彩强度，显示出物体所具有的外观。因此，需要开发从图形到图像的转换算法，实现图形到图像的转换，最终在屏幕上呈现。

围绕从图形输入到图像显示的流水线，本章介绍相关基础知识，重点介绍流水线处理中的几何变换，如模型变换、视点变换、投影变换等，以及图元到像素的光栅化处理，如剔除、填充等操作。此外，本章还简要介绍专门执行图形绘制的计算机硬件——显卡。

2.1 图形的显示

图形显示时需要转换为图像，然后在计算机屏幕等终端设备上进行显示，才能被人眼所看到。从图形到图像的转换过程是十分复杂的，涉及形状、颜色、亮度、透明度等属性的变化。简单地讲，场景模型经过一系列的几何变换映射到二维图像，确定图元对应的图像像素坐标位置，其中涉及各种物理坐标系的转换(见 2.2 节)。图元在转换成像素时，则需要确定指定分辨率下的哪些像素被覆盖，以及以何种 R、G、B 颜色分量的组合进行色彩显示，也就是光栅化处理(见 2.3 节)。因此，如图 2.1 所示，从图形到图像显示的整个过程，可以看作是一条数据处理的流水线，经历了对几何、颜色等的一系列处理步骤。其中，几何变换和光栅化是最重要的两个组成部分。

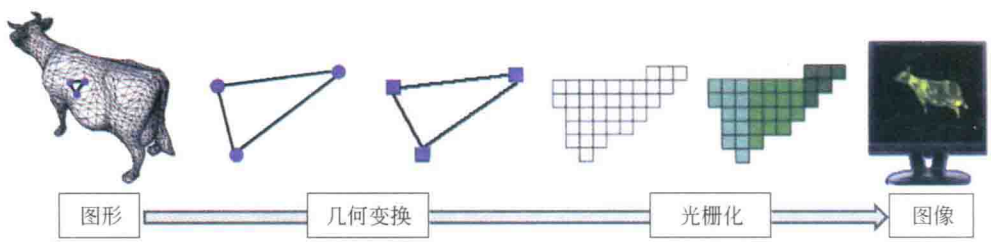


图 2.1 图形流水线的基本步骤

以图 2.1 中的图形流水线为例，输入图形通常是以三维空间中的点、线、三角形面片作为图元组成的三维模型。通过几何变换，将组成模型的三角形面片从三维空间变换到指定的位置，相应的三角形顶点也被映射到指定视角下的投影平面，从而获得顶点对应的图像坐标，并进一步转换成显示终端上指定窗口的屏幕坐标。这样，就完成了三维模型从图元到像素的坐标转换。通过光栅化处理，确定顶点、边、三角形面片等图元对应的屏幕像素，也就是哪些像素会被组成模型的图元所覆盖。同时，根据光照、材质、纹理、遮挡等因素，计算该像素的 RGB 值，作为模型在显示时的颜色。经过上述过程，图形最终转换为二维屏幕上能显示的图像。

实际的计算机图形学系统中，图形流水线是非常复杂的。以 OpenGL 为例，这是一种跨平台、开放的专业图形程序接口，已成为业界公认的图形标准之一，能够满足对不同模型的不同显示效果的图形绘制。目前的最新版本是 OpenGL 4.6。它在以往图形流水线基础上增加了新的 GPU 编程接口（部分内容参考第 10 章）。图 2.2 展示了 OpenGL 的图形流水线处理过程。事实上，流水线上开放的程序接口定义了从图形输入到图像输出所经历的典型操作步骤。这些步骤按照操作对象大体可以分为两条主线：处理图元元素的几何操作和处理图像元素的像素操作。

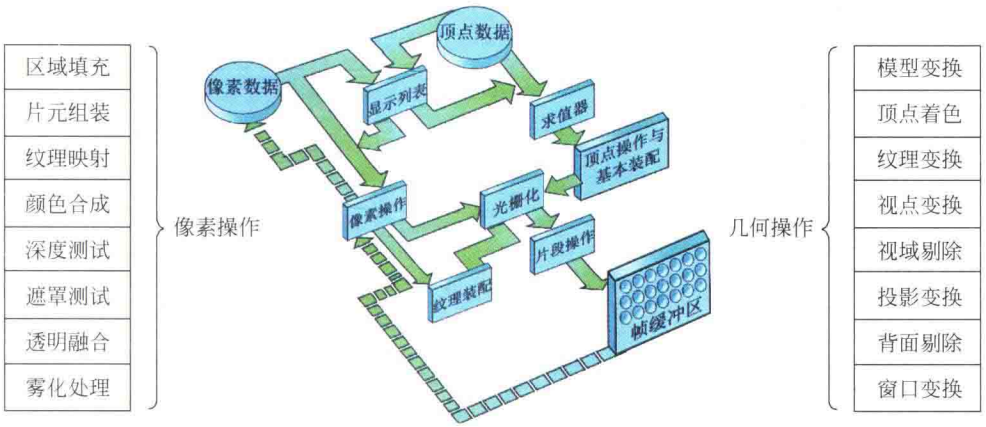


图 2.2 OpenGL 图形流水线

流水线上的几何操作从图形输入开始，通过模型变换、视点变换、投影变换、窗口变换等，将图元从局部坐标系映射到屏幕坐标系。这个过程根据光源位置、材质属性、纹理属性、遮挡属性、透明属性等，设置组成图形的各个图元的绘制参数。同时，对于模型的可见性等属性进行判断，形成在屏幕窗口显示时的各种限制条件。像素操作则是根据几何操作的结果，处理与图元相对应的屏幕像素，包括填充每个图元覆盖的区域，结合纹理进行像素颜色填充。这个过程对来自不同图元的填充像素进行深度测试、遮罩测试等操作，以及透明融合、雾化处理等特殊效果的绘制。最终，图形在屏幕上绘制成二维图像进行显示。

2.2 几何变换

图形流水线上的几何操作主要通过各种形式的几何变换来实现。数学上，每一种几何变换操作都是通过矩阵和矩阵或向量的代数运算进行计算的。这些几何变换将输入的三维模型上的点，从局部坐标系变换到屏幕坐标系，实质上是建立模型图元到屏幕像素的点-点映射。图形流水线上的几何变换主要包括模型变换、视点变换、投影变换和窗口变换，涉及的坐标系包括物体坐标系、世界坐标系、眼睛坐标系、屏幕坐标系等。图 2.3 展示了主要的几何变换及相应的物理坐标系变化，其中局部坐标系、世界坐标系和眼睛坐标系是三维空间坐标系，而图像坐标系和屏幕坐标系则是二维平面坐标系。事实上，这些坐标系提供了在不同视角下对图形进行表示的基本空间。

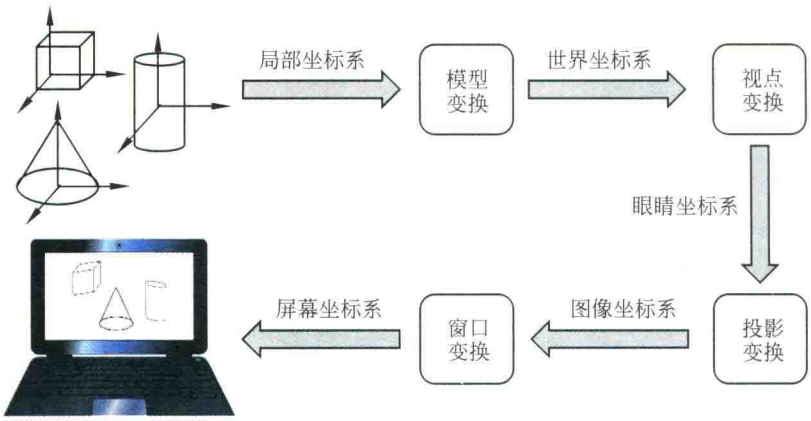


图 2.3 图形流水线中的几何变换

一般而言，三维空间坐标系由 x、y、z 三个方向彼此正交的坐标轴和原点 O 构成。三维空间中的点 $\boldsymbol{p}$ 具有三个坐标分量，记为 $\boldsymbol{p}=(x,y,z)$。为了统一表示所有的几何变换，通常采用齐次坐标(Homogeneous coordinates)表示三维空间的点，采用向量形式可以记为 $\tilde{\boldsymbol{p}}=[x,y,z,1]^{\mathrm{T}}$。在齐次坐标系下，相差一个共同尺度因子的两个向量表示三维空间中的同一个点，也就是 $[x,y,z,1]^{\mathrm{T}}\sim[\lambda x,\lambda y,\lambda z,\lambda]^{\mathrm{T}}$。

这样，三维空间中模型的几何变换可以采用 4×4 矩阵 $\boldsymbol{M}$ 与 4×1 齐次坐标向量的乘积运算进行表示和计算，记为如下公式：

$$\boldsymbol{M}\cdot\tilde{\boldsymbol{p}}=\begin{bmatrix}m_{11}&m_{12}&m_{13}&m_{14}\\m_{21}&m_{22}&m_{23}&m_{24}\\m_{31}&m_{32}&m_{33}&m_{34}\\m_{41}&m_{42}&m_{43}&m_{44}\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}=\begin{bmatrix}m_{11}x+m_{12}y+m_{13}z+m_{14}\\m_{21}x+m_{22}y+m_{23}z+m_{24}\\m_{31}x+m_{32}y+m_{33}z+m_{34}\\m_{41}x+m_{42}y+m_{43}z+m_{44}\end{bmatrix}\tag{2.1}$$

那么，三维空间中的点 $\boldsymbol{p}$ 在变换之后得到点 $\boldsymbol{p}'=(x',y',z')$，其三个坐标分量具有如下形式：

$$\begin{cases} x' = (m_{11}x + m_{12}y + m_{13}z + m_{14})/(m_{41}x + m_{42}y + m_{43}z + m_{44}) \\ y' = (m_{21}x + m_{22}y + m_{23}z + m_{24})/(m_{41}x + m_{42}y + m_{43}z + m_{44}) \\ z' = (m_{31}x + m_{32}y + m_{33}z + m_{34})/(m_{41}x + m_{42}y + m_{43}z + m_{44}) \end{cases}$$

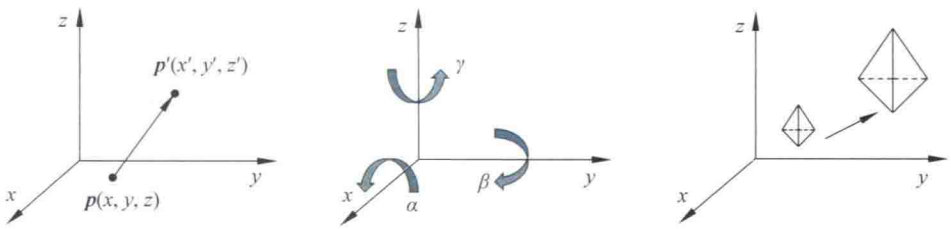
类似地，二维平面坐标系下的点 $\boldsymbol{q}=(x,y)$，也可以采用这种齐次坐标向量表示形式，记为 $\tilde{\boldsymbol{q}}=[x,y,1]^{\mathrm{T}}$，其中 x 和 y 是分别对应于 x 轴和 y 轴的坐标分量。这种采用齐次坐标向量形式的好处，是可以对图形流水线中的各种几何变换进行统一表示和计算，从而方便图形流水线上对图元的各种几何操作。

接下来，具体介绍图形流水线上用到的几种典型几何变换。

2.2.1 模型变换

模型变换将物体从局部坐标系变换到世界坐标系。局部坐标系是以物体为中心构建的一个坐标系，方便用户对单个物体进行各种操作。这是因为人们在与真实物体进行交互时，往往先不考虑该物体和其他物体的位置关系，而是习惯在一个局部的坐标系中对物体模型进行几何设计。然后，当每个物体的几何模型设计完成后，再按照物体之间的相对位置关系转换到同一个坐标系下进行统一表示。这个同一坐标系便是世界坐标系。局部坐标系和世界坐标系都是三维坐标系，通过使用不同模型变换描述物体模型的几何性质及相对位置。

如图 2.4 所示，模型变换主要涉及平移、旋转、缩放等几何操作，都是从三维空间映射到三维空间的几何变换。这里采用三维空间的齐次坐标形式，所有的几何变换就可以采用4×4 的矩阵进行代数表示。模型变换的结果可以借助矩阵和矩阵或向量的运算获得。



(a) 平移变换　　(b) 旋转变换　　(c) 比例变换

图 2.4　三种典型的模型变换

1. 平移变换

平移变换是沿着一个固定的方向将点 $\boldsymbol{p}$ 移动一个指定的距离，得到新的空间位置坐标 $\boldsymbol{p}'=(x',y',z')$。平移变换的矩阵形式如公式(2.2)所示，那么该变换表示为 $\tilde{\boldsymbol{p}}'=\boldsymbol{T}\cdot\tilde{\boldsymbol{p}}$。公式(2.2)中的向量 $[t_x,t_y,t_z]$ 代表了在三维空间从点 $\boldsymbol{p}$ 到点 $\boldsymbol{p}'$ 的位移向量。如图 2.4(a)所示平移变换仅改变物体模型的空间位置，而不会改变其形状和姿态。

$$\boldsymbol{T}=\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

2. 旋转变换

旋转变换是能够保持至少一个点的几何位置不变的刚性变换。在二维平面上，该点常被称为旋转中心；而在三维空间中，该点是旋转中心或旋转轴上的点。与二维平面上的旋转不同，三维空间中的旋转变换更加复杂，表示形式也不是唯一的。常用的三维旋转变换表示形式有欧拉角、四元数等。欧拉角的形式是分别用绕 x、y、z 三个轴的旋转角度 α、β、γ 表示。这三个角也分别称为俯仰角(Pitch)、偏航角(Yaw)和翻滚角(Roll)。该表示形式具有更直观的几何意义。对应的旋转变换矩阵 $\boldsymbol{R}_x$、$\boldsymbol{R}_y$、$\boldsymbol{R}_z$ 如公式(2.3)所示。那么，最后的旋转变换就是这三个绕轴旋转变换矩阵的乘积结果，也称为欧拉变换。

$$\boldsymbol{R}_x=\begin{bmatrix}1&0&0&0\\0&\cos\alpha&-\sin\alpha&0\\0&\sin\alpha&\cos\alpha&0\\0&0&0&1\end{bmatrix}\quad \boldsymbol{R}_y=\begin{bmatrix}\cos\beta&0&\sin\beta&0\\0&1&0&0\\-\sin\beta&0&\cos\beta&0\\0&0&0&1\end{bmatrix}\quad \boldsymbol{R}_z=\begin{bmatrix}\cos\gamma&-\sin\gamma&0&0\\\sin\gamma&\cos\gamma&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \tag{2.3}$$

需要注意的是，欧拉角表示的旋转和绕三个坐标轴旋转的先后顺序有关，也就是说对应的绕轴旋转矩阵不具备交换性质，即 $\boldsymbol{R}_x\cdot\boldsymbol{R}_y\neq\boldsymbol{R}_y\cdot\boldsymbol{R}_x$。因此，欧拉变换必须严格限定绕轴旋转的先后顺序。此外，使用欧拉角表示的三维空间内旋转还存在万向节死锁(Gimbal lock)现象。这种现象发生在两个绕轴的旋转重合时，此时就会失去一个旋转自由度，从而导致多组欧拉角取值对应同一个位置。在这种情形下，使用四元数表示三维空间的旋转是更佳的选择。但是由于矩阵表示和变换计算的便捷性，目前的图形流水线中仍以欧拉矩阵作为主要的旋转变换表示形式。

3. 比例变换

比例变换是按照一定的比例放大或者缩小物体的几何模型，记为 $\tilde{\boldsymbol{p}}'=\boldsymbol{S}\cdot\tilde{\boldsymbol{p}}$。缩放变换对应的矩阵形式如公式(2.4)所示。其中，对角线上的数值 s_x、s_y 和 s_z，分别代表了沿 x、y、z 轴三个方向的缩放比例。当 $s_x=s_y=s_z$ 时，模型发生了等比例的缩放，也就是相似变换(如图 2.4(c)所示)；否则，模型在各个方向的缩放比例不同，会产生形状的变化。

$$\boldsymbol{S}=\begin{bmatrix}s_x&0&0&0\\0&s_y&0&0\\0&0&s_z&0\\0&0&0&1\end{bmatrix} \tag{2.4}$$

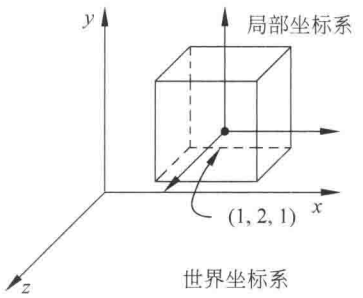
此外，还有剪切变换、镜像变换等形式的几何变换，都可以通过 4×4 矩阵表示。通过这些几何变换，可以将不同局部坐标系下的物体模型统一到同一个世界坐标系下来表示。

例题 2-1　计算正方体从局部坐标系到世界坐标系的模型变换矩阵。

问题： 如下页图所示，假设单位正方体所在局部坐标系的原点位于该正方体的中心位置，而且在世界坐标系下的坐标为(1,2,1)，那么该单位正方体的 8 个顶点 A、B、C、D、E、F、G、H 在世界坐标系下的坐标分别是什么？

解答： 单位正方体的 8 个顶点在以正方体中心为原点的局部坐标系下的坐标分别为

$$\begin{cases} A=(-0.5,0.5,0.5) \\ B=(0.5,0.5,0.5) \\ C=(0.5,0.5,-0.5) \\ D=(-0.5,0.5,-0.5) \\ E=(-0.5,-0.5,0.5) \\ F=(0.5,-0.5,0.5) \\ G=(0.5,-0.5,-0.5) \\ H=(-0.5,-0.5,-0.5) \end{cases}$$



从局部坐标系到世界坐标系是平移变换，而且该变换将(0,0,0)变为(1,2,1)。因此，该平移变换矩阵表示为

$$\boldsymbol{M}_{l2w}=\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

那么，根据齐次坐标向量表示，8个顶点在平移变换后的齐次坐标分别计算为 $\boldsymbol{M}_{l2w}\cdot[A,1]$、$\boldsymbol{M}_{l2w}\cdot[B,1]$、$\boldsymbol{M}_{l2w}\cdot[C,1]$、$\boldsymbol{M}_{l2w}\cdot[D,1]$、$\boldsymbol{M}_{l2w}\cdot[E,1]$、$\boldsymbol{M}_{l2w}\cdot[F,1]$、$\boldsymbol{M}_{l2w}\cdot[G,1]$、$\boldsymbol{M}_{l2w}\cdot[H,1]$。进而，它们在世界坐标系下的坐标分别为

$$\begin{cases} A_w=(0.5,2.5,1.5) \\ B_w=(1.5,2.5,1.5) \\ C_w=(1.5,2.5,0.5) \\ D_w=(0.5,2.5,0.5) \\ E_w=(0.5,1.5,1.5) \\ F_w=(1.5,1.5,1.5) \\ G_w=(1.5,1.5,0.5) \\ H_w=(0.5,1.5,0.5) \end{cases}$$

□

2.2.2 视点变换

人借助眼睛所观看到的三维世界中的物体，是在确定的人眼位置、方向和视场范围内形成的视网膜投影成像。因此，三维图形在转换为二维图像显示时，也需要将场景中的物体模型转换到以人眼为中心的坐标系下进行表示，这就是视点变换。在图形流水线中，视点变换需要计算一个从世界坐标系到眼睛坐标系的变换矩阵 $\boldsymbol{M}_{w2e}$。该变换将三维空间中世界坐标系下的物体模型转换到以人眼为原点、视线朝向等为坐标轴的三维眼睛坐标系，从而建立人眼视角下的模型表示。

假设眼睛在世界坐标系中的位置坐标是 $\boldsymbol{e}=(e_x,e_y,e_z)$，眼睛坐标系的三个主轴方向分别是 $\boldsymbol{n}=(n_x,n_y,n_z)$、$\boldsymbol{v}=(v_x,v_y,v_z)$ 和 $\boldsymbol{u}=(u_x,u_y,u_z)$。其中，$\boldsymbol{n}$ 一般设置为眼睛观看方向的反方向；$\boldsymbol{v}$ 是眼睛的正朝向，通常和世界坐标系的 $\boldsymbol{y}$ 轴朝向一致，且与 $\boldsymbol{n}$ 垂直；$\boldsymbol{u}$ 则是垂直于 $\boldsymbol{n}$ 和 $\boldsymbol{v}$ 张成平面的向量，也就是 $\boldsymbol{u}=\boldsymbol{v}\times\boldsymbol{n}$。这样，就建立了以眼睛作为原点的眼

睛坐标系。那么，视点变换就是要将世界坐标系的原点 $\boldsymbol{o}=(0,0,0)$ 变换到新的位置 $\boldsymbol{e}$，将 $\boldsymbol{x}=(1,0,0)$、$\boldsymbol{y}=(0,1,0)$ 和 $\boldsymbol{z}=(0,0,1)$ 三个坐标轴分别变换到 $\boldsymbol{u}$、$\boldsymbol{v}$ 和 $\boldsymbol{n}$ 所表示的向量，代表眼睛坐标系下的 $\boldsymbol{u}$、$\boldsymbol{v}$、$\boldsymbol{n}$ 三个坐标轴。

为了便于计算世界坐标系到眼睛坐标系的变换矩阵，通常是首先根据上述坐标系的对应关系计算其逆变换 $\boldsymbol{M}_{e2w}$，也就是眼睛坐标系到世界坐标系的几何变换，然后利用矩阵求逆，得到变换矩阵 $\boldsymbol{M}_{w2e}$。通过简单的矩阵和向量运算 $[\boldsymbol{e},1]^{\mathrm{T}}=\boldsymbol{M}_{e2w}\cdot[o,1]^{\mathrm{T}}$，可以得到矩阵 $\boldsymbol{M}_{e2w}$ 的最后一列就是眼睛位置在世界坐标系下的位置坐标。

同理，将眼睛坐标系的三个坐标轴映射为世界坐标系下的三个坐标轴，就可以得到矩阵 $\boldsymbol{M}_{e2w}$ 的其他值。因此，该矩阵满足公式(2.5)的等式关系：

$$\begin{cases}\boldsymbol{M}_{e2w}\cdot\widetilde{\boldsymbol{o}}=\widetilde{\boldsymbol{e}}\\ \boldsymbol{M}_{e2w}\cdot\widetilde{\boldsymbol{x}}=\widetilde{\boldsymbol{u}}'\\ \boldsymbol{M}_{e2w}\cdot\widetilde{\boldsymbol{y}}=\widetilde{\boldsymbol{v}}'\\ \boldsymbol{M}_{e2w}\cdot\widetilde{\boldsymbol{z}}=\widetilde{\boldsymbol{n}}'\end{cases}\tag{2.5}$$

上述公式中的第二个等式是将眼睛坐标系下的 u 轴上的点(1,0,0)映射到世界坐标系下 x 轴上的点，也就是 $\boldsymbol{u}'=(e_x+u_x,e_y+u_y,e_z+u_z)$，从而可以得到 $\boldsymbol{M}_{e2w}$ 的第一列就是向量 $\boldsymbol{u}$。第三个等式是将眼睛坐标系下的 v 轴上的点(0,1,0)映射到世界坐标系下 y 轴上的点，也就是 $\boldsymbol{v}'=(e_x+v_x,e_y+v_y,e_z+v_z)$，从而可以得到 $\boldsymbol{M}_{e2w}$ 的第二列就是向量 $\boldsymbol{v}$。第四个等式是将眼睛坐标系下的 n 轴上的点(0,0,1)映射到世界坐标系下 z 轴上的点，也就是 $\boldsymbol{n}'=(e_x+n_x,e_y+n_y,e_z+n_z)$，从而可以得到 $\boldsymbol{M}_{e2w}$ 的第三列就是向量 $\boldsymbol{n}$。因此，得到该视点变换对应的矩阵表示如下所示：

$$\boldsymbol{M}_{e2w}=\begin{bmatrix}u_x & v_x & n_x & e_x\\ u_y & v_y & n_y & e_y\\ u_z & v_z & n_z & e_z\\ 0 & 0 & 0 & 1\end{bmatrix}\tag{2.6}$$

它反映了在齐次坐标系下，从眼睛坐标系到世界坐标系的几何变换。

接下来计算 $\boldsymbol{M}_{e2w}$ 的逆矩阵，可以得到如下表示的世界坐标系到眼睛坐标系的变换矩阵 $\boldsymbol{M}_{w2e}$：

$$\boldsymbol{M}_{w2e}=\begin{bmatrix}u_x & u_y & u_z & -\boldsymbol{u}\cdot\boldsymbol{e}\\ v_x & v_y & v_y & -\boldsymbol{v}\cdot\boldsymbol{e}\\ n_x & n_y & n_z & -\boldsymbol{n}\cdot\boldsymbol{e}\\ 0 & 0 & 0 & 1\end{bmatrix}\tag{2.7}$$

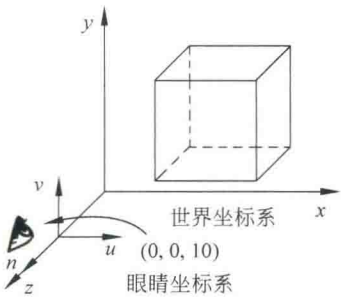
通过公式(2.7)所示的变换，就可以将世界坐标系下的物体模型转换到眼睛坐标系，实现在任意视点位置沿着视线朝向对模型的观看。

例题 2-2　计算单位正方体从世界坐标系到眼睛坐标系的视点变换矩阵。

问题：假设单位正方体位于例题 2-2 所示的世界坐标系，而眼睛所处位置的世界坐标为(0,0,10)，并且具有如右图所示的 u、v、n 坐标轴朝向，那么该单位正方体的 8 个顶点 A、B、C、D、E、F、G、H 在眼睛坐标系下的坐标分别是什么？

解答：如下图所示，世界坐标系和眼睛坐标系之间可以通过平移变换进行转换。该平移变换的矩阵是

$$\boldsymbol{M}_{w2e}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & -10\\ 0 & 0 & 0 & 1\end{bmatrix}$$



再结合例题 2-1 中从局部坐标系到世界坐标系的变换，可以得到从局部坐标系到眼睛坐标系的变换是：

$$\boldsymbol{M}_{l2e}=\boldsymbol{M}_{l2w}\cdot\boldsymbol{M}_{w2e}=\begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 1 & 0 & 2\\ 0 & 0 & 1 & -9\\ 0 & 0 & 0 & 1\end{bmatrix}$$

那么单位正方体的八个顶点的齐次坐标分别计算为 $\boldsymbol{M}_{l2e}\cdot[A,1]$、$\boldsymbol{M}_{l2e}\cdot[B,1]$、$\boldsymbol{M}_{l2e}\cdot[C,1]$、$\boldsymbol{M}_{l2e}\cdot[D,1]$、$\boldsymbol{M}_{l2e}\cdot[E,1]$、$\boldsymbol{M}_{l2e}\cdot[F,1]$、$\boldsymbol{M}_{l2e}\cdot[G,1]$、$\boldsymbol{M}_{l2e}\cdot[H,1]$。进而，它们在眼睛坐标系下的坐标分别是：

$$\begin{cases}A_e=(0.5,2.5,-8.5)\\ B_e=(1.5,2.5,-8.5)\\ C_e=(1.5,2.5,-9.5)\\ D_e=(0.5,2.5,-9.5)\\ E_e=(0.5,1.5,-8.5)\\ F_e=(1.5,1.5,-8.5)\\ G_e=(1.5,1.5,-9.5)\\ H_e=(0.5,1.5,-9.5)\end{cases}$$

□

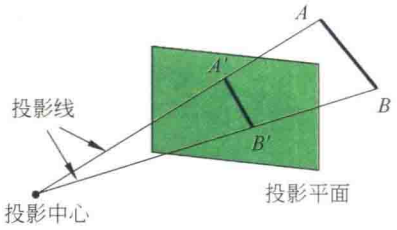
2.2.3 投影变换

物体模型从世界坐标系转换到眼睛坐标系后，需要通过投影变换产生二维图像，也就是将眼睛坐标系中的物体模型投影到二维图像坐标系。简单地讲，投影变换的结果可以看作从投影中心发出的射线，穿过场景中物体模型表面的每一个点，然后与投影平面相交所形成的图像。因此，投影变换主要涉及三个几何概念：投影中心、投影平面和投影线（如图 2.5(a)所示）。在图形流水线中，投影变换包括透视投影和正交投影两种投影方式。

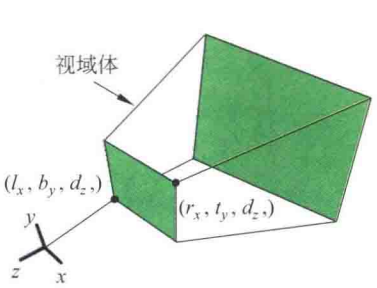
1. 透视投影

假设投影中心到投影平面的距离是有限的。按照投影后图像中消隐点的个数可以分为单点透视、两点透视和三点透视。这样使二维图像能够呈现三维空间的立体感，达到了在二维屏幕上更好地呈现三维模型的目的。而在实际的图形流水线中，透视投影主要采用单点透视。

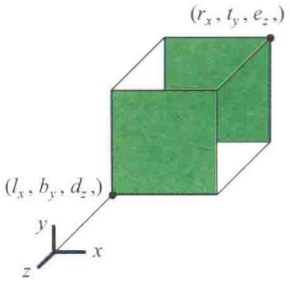
透视投影本质上也是一种几何变换，仍然可以通过 4×4 的矩阵表示。为了方便实际使用，透视投影通常借助如图 2.5(b)所示的两个平行平面组成的视域体来定义。这两个



(a) 投影变换



(b) 透视变换



(c) 正交变换

图 2.5 投影变换

分别位于前后位置的平面称为裁剪面。它们到眼睛的距离分别记为 d_z 和 f_z，并且满足 $d_z \leqslant f_z$。其中，靠近人眼的称为近裁剪面，通过左下顶点 (l_x, b_y, d_z) 和右上顶点 (r_x, t_y, d_z) 定义的矩形区域来表示；远离人眼的称为远裁剪面，借助经过近裁剪面矩形顶点的投影线和远裁剪面的交点所围成的矩形来表示。该视域体一方面定义了投影成像所在的近裁剪面，也就是三维模型最终投影到该裁剪面上形成二维图像；另一方面，它也定义了场景的可见范围，也就是位于近裁剪面和远裁剪面之间的物体模型，才能够最终在屏幕上呈现。

从眼睛坐标系到近裁剪面上图像坐标系的投影变换 $\boldsymbol{P}_{e2i}$，可以写成如下的矩阵形式：

$$\boldsymbol{P}_{e2i} = \begin{bmatrix} \dfrac{2d_z}{r_x - l_x} & 0 & \dfrac{r_x + l_x}{r_x - l_x} & 0 \\ 0 & \dfrac{2d_z}{t_y - b_y} & \dfrac{t_y + b_y}{t_y - b_y} & 0 \\ 0 & 0 & -\dfrac{f_z + d_z}{f_z - d_z} & -\dfrac{2f_z d_z}{f_z - d_z} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{2.8}$$

此时，眼睛坐标系下的点 $\tilde{\boldsymbol{p}} = [x, y, z, 1]^{\mathrm{T}}$ 就变换为 $\tilde{\boldsymbol{p}}' = [x', y', z', 1]^{\mathrm{T}}$，其三个坐标分量具有如下形式的表达式：

$$\begin{cases} x' = \left(\dfrac{2d_z}{r_x - l_x} x + \dfrac{r_x + l_x}{r_x - l_x} z \right) \Big/ (-z) \\ y' = \left(\dfrac{2d_z}{t_y - b_y} y + \dfrac{t_y + b_y}{t_y - b_y} z \right) \Big/ (-z) \\ z' = \left(-\dfrac{f_z + d_z}{f_z - d_z} z - \dfrac{2f_z d_z}{f_z - d_z} \right) \Big/ (-z) \end{cases} \tag{2.9}$$

最终，$\boldsymbol{q}=(x',y')$是在近裁剪面上投影所对应的图像坐标，而z'作为该点的深度值会进一步做深度测试等操作。

事实上，公式(2.8)定义的投影变换将视域体变换为规则的正方体，称为标准设备坐标系。该正方体在眼睛坐标系下是以原点为中心，同时以(−1，−1，−1)作为最左下顶点、(1，1，1)作为最右上顶点。因此，变换后的图像坐标分量x'和y'应当位于区间[−1，1]的范围之内。

2. 正视投影

这种投影又称为正射投影或平行投影。假设投影中心到成像平面的距离为无穷远，也就是投影中心的位置在距离近裁剪面无穷远处，那么从投影中心发出的投影线是平行的(如图 2.5(c)所示)。因此，正视投影的视域体的近裁剪面和远裁剪面大小相同，导致投影图像也是相同大小，不会产生“近大远小”的视觉效果。相比于透视投影，正视投影变换的矩阵表示较为简单，具有如下的矩阵表示形式：

$$\boldsymbol{P}_{e2i}^{\text{otho}}=\begin{bmatrix}\dfrac{2}{r_x-l_x} & 0 & 0 & -\dfrac{r_x-l_x}{r_x-l_x}\\ 0 & \dfrac{2}{t_y-b_y} & 0 & -\dfrac{t_y+b_y}{t_y-b_y}\\ 0 & 0 & -\dfrac{2}{f_z-d_z} & -\dfrac{f_z+d_z}{f_z-d_z}\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{2.10}$$

正视投影也是将两个裁剪面围成的视域体变换到标准设备坐标系。从成像效果的角度来看，透视投影所获得的图像与人类视觉系统更为相似，都具有“透视缩短”的效果，也就是物体的大小与投影中心的距离成反比，远到极点即为消失，称为灭点。这使得投影后的图像具有“近大远小”的视觉效果，而平行于投影面的直线夹角保持不变。正视投影则不具备透视效果，缺乏符合人类视觉感观的成像效果，但在建筑蓝图绘制和 CAD 图纸设计等方面有着专门的应用。这些行业往往要求投影后的物体尺寸及相互间的角度不变，从而方便施工或制造时物体比例大小正确。

2.2.4 窗口变换

物体模型经过投影变换后变成二维图像，需要进一步映射到屏幕上的窗口进行显示。这个过程就是窗口变换，它将物体模型在视域体近裁剪面上的二维图像坐标变换为二维屏幕坐标，从而确定图形转换为图像进行显示时图元所覆盖的像素位置及区域。屏幕上的窗口表示为规整的矩形，其左下顶点是(v_x,v_y)，宽和高分别是w和h，那么窗口变换$\boldsymbol{V}_{i2s}$的矩阵表示如公式(2.11)所示。

$$\boldsymbol{V}_{i2s}=\begin{bmatrix}\dfrac{w}{2} & 0 & 0 & v_x+\dfrac{w}{2}\\ 0 & \dfrac{h}{2} & 0 & v_y+\dfrac{h}{2}\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{2.11}$$

如图 2.6 所示，该窗口变换把标准设备坐标系下的前裁剪面映射到屏幕窗口，也就是将坐标为(−1，−1)的点映射为坐标为(v_x, v_y)的点，将坐标为(1，1)的点映射为坐标为(v_x+w, v_y+h)的点。这样通过窗口变换，三维模型对应的图形就变换为屏幕上的二维图像，可以借助像素颜色进行正确的显示。

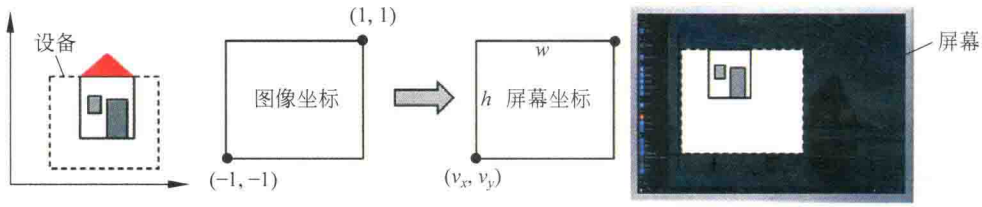


图 2.6 窗口变换

2.3 光栅化处理

光栅化是将几何图元转换成光栅图像(点阵或像素)，从而能够在屏幕上按照指定的像素 RGB 颜色进行输出和显示的过程。几何变换的图元经过光栅化处理，实现从二维/三维图形到二维图像的转换。在图形流水线中，光栅化处理主要包括简单图元生成、多边形填充、剔除、可见性判断等操作步骤，最终生成可在屏幕上显示的图像。

2.3.1 简单图元生成

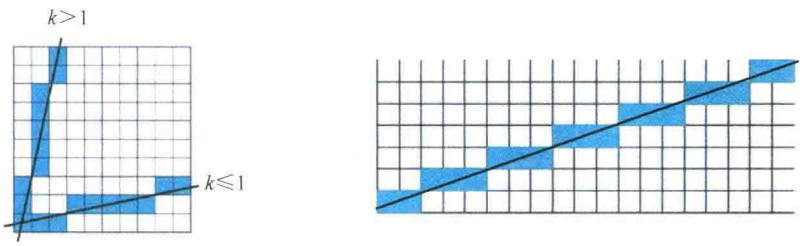
复杂的图形是由基本的图元组合而成的，因此首先需要解决如何将图元变为像素集合。例如一条直线，数学上可以通过两个顶点的连线来定义。但在屏幕上显示的时候，直线按照一定的长度和宽度来绘制，需要设计直线光栅化算法来计算哪些像素在直线显示时被覆盖。这涉及直线的画线算法和裁剪算法。典型的画线算法有数字微分分析(DDA)算法、Bresenham 算法等。虽然这些算法产生于计算机图形学发展的早期阶段，其软/硬件实现都已经非常成熟，但仍是进行图形绘制最经典和基本的算法。

1. DDA 算法

DDA 算法模拟微分方程的数值求解过程，将直线表示为满足微分方程 $\mathrm{d}y/\mathrm{d}x=k$ 的解。该方程对应的有限差分形式为 $\Delta y=k\cdot\Delta x$。这里 k 表示直线的斜率，反映了像素的 y 坐标分量的变化 Δy，随另外一个 x 坐标分量变化 Δx 的情况。由于屏幕上的像素坐标都是整数，需要根据直线的斜率对最佳的 Δx 和 Δy 的取值进行合适的取舍处理。如图 2.7(a)所示的第一象限中的线段，当斜率小于 1 时，x 坐标轴方向为主步进方向，假定 x 值增量为 $\Delta x=1$，那么最佳的 y 值增量满足 $\Delta y=k$；当斜率大于 1 时，y 坐标轴方向为主步进方向，需要将 x 和 y 交换，使得 y 值增量为 $\Delta y=1$，那么最佳的 x 值增量满足 $\Delta x=1/k$。这样处理过的直线，能够保证其在屏幕上显示时，所覆盖的像素是连接在一起的，而不至于发生截断。

例题 2-3 DDA 画线算法绘制线段。

问题：写出采用 DDA 算法进行线段绘制的伪代码。



(a) DDA算法　　(b) Bresenham算法

图 2.7 DDA 画线算法与 Bresenham 画线算法

解答：假设 p1 和 p2 是线段的两个端点，绘制该线段的伪代码是

```
function DDADrawLine (Point p1, Point p2)
    k ← calculate_slope (p1, p2)                    /* 计算斜率 */
    if abs (k) <=1 then                             /* x 方向占优的线段 */
        (x, y) ← get_left_point (p1, p2)
        (x_r, y_r) ← get_right_point (p1, p2)
        repeat
            draw_pixel (x, y)
            x ← x +1
            y ← y +k
        until x ==x_r end
    else                                            /* y 方向占优的线段 */
        (x, y) ← get_buttom_point (p1, p2)
        (x_t, y_t) ← get_top_point (p1, p2)
        repeat
            draw_pixel (x, y)
            x ← x +1/k
            y ← y +1
        until y ==y_t end
    end if
end function
```

□

2. Bresenham 算法

Bresenham 算法是一种根据数值浮点运算规律加速直线绘制的方法。在绘制直线时，DDA 算法每生成一个像素都要对浮点数进行加减运算，整条线生成的速度较慢。1962 年，IBM 公司的 Jack Bresenham 发明了 Bresenham 画线算法。以斜率满足 $|k|\leqslant 1$ 的直线为例，假设直线通过 (x_1, y_1) 和 (x_2, y_2) 两点，同时记 $\Delta y = y_2 - y_1$，$\Delta x = x_2 - x_1$，那么 $k = \Delta y / \Delta x$。该算法将 DDA 算法中的浮点数增量 k，改为根据误差函数 $e = 2\Delta y - \Delta x$ 的符号来确定整数增量为 0 或 1。如图 2.7(b)所示，当 $e > 0$ 时，y 坐标轴方向的增量为 1，同时误差函数更新为 $e \leftarrow e - 2\Delta x$；否则，$y$ 坐标轴方向的增量为 0，x 坐标轴方向的增量为 1。由此可见，Bresenham 算法仅仅使用整数增量计算，计算量是非常小的，很容易采用硬件来实现。此外，采用 Bresenham 算法也可以用来绘制圆形、椭圆等其他简单图元。

例题 2-4　Bresenham 算法绘制线段。

问题：写出采用 Bresenham 算法进行线段绘制的伪代码。

解答：假设 p1 和 p2 是线段的两个端点，绘制该线段的伪代码是

```
function BresenhamDrawLine (Point p1, Point p2)
    dx ← p2.x -p1.x
    dy ← p2.y -p1.y
    e ← 2 * dy -dx                          /*计算误差函数*/
    (x, y) ← p1.index                       /*初始化线段起点*/
    repeat
        draw_pixel (x, y)
        if e <0 then                        /*更新误差函数*/
            e ← e +2 * dy
        else
            e ← e +2 * (dy -dx)
            y++
        end if
    until x++==p2.x end
end function
```

□

2.3.2　多边形填充

复杂模型的表面通常由简单的封闭多边形面片组合而成，例如三角面片相邻的边拼接形成复杂三角网格。这些多边形在进行绘制时，也需要对多边形内部进行光栅化处理，才能得到一个表面封闭的模型。这个过程称为区域填充。这类算法实质上是要确定屏幕上哪些像素位于多边形内部，并采用相应的颜色对这些内部像素进行绘制来填充多边形。

最简单的填充算法是逐像素判断是否在多边形内部，但是效率很低，不利于复杂模型的快速绘制。扫描转换是图形流水线中经常使用的区域填充算法，其基本思想是通过逐行扫描来记录每一条扫描线和多边形的边的交点。然后，基于扫描区域的连贯性完成多边形内部区域的像素填充。

在进行扫描转换时，通常选取平行于 x 轴的水平线作为扫描线。从多边形最底端开始，也就是满足 $y=y_{min}$ 的行，自下而上逐行扫描。对于每一行，利用多边形边界的直线方程得到与扫描线的交点，并按照从左到右的顺序对交点进行编号（如图 2.8 所示）。

然后，利用交点数量的奇偶规则判断多边形边界所包围的内部区域：奇数表示在多边形内部，偶数表示在多边形外部。例如，在图 2.8 所示的例子中，编号为 2 和 3 的点是产生在第 1 个（奇数）交点之后，属于多边形内部；编号为 5、6、7 的点是产生在第 2 个交点（偶数）之后，属于多边形外

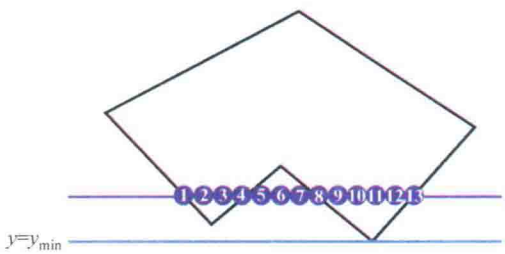


图 2.8　多边形扫描线填充算法示意图

部；编号为9、10、11、12的点产生在第3个交点(奇数)之后，属于多边形内部。

为了降低扫描算法的空间复杂度，通常采用链表数据结构记录交点位置，并使用插入排序方法对扫描像素的 x 坐标排序，从而提高计算效率。

例题 2-5 用扫描线算法绘制多边形。

问题：写出采用扫描线算法绘制多边形的伪代码。

解答：假设预定义的多边形为 poly，通过扫描线填充进行绘制的伪代码是

```
function PolyScanLine (Polygon poly)
    for each y in get_lines (poly) do          /* 沿着 y 方向进行扫描 */
        ps ← get_crossover_points (poly, y)    /* 计算扫描线和边界交点 */
        number_points (ps)
        is ← get_inner_sections (ps)           /* 计算内部像素 */
        fill_poly (is)                         /* 填充内部像素 */
    end for
end function
```

□

2.3.3 剔除

在进行光栅化时，需要确定场景中哪些物体模型在经过几何变换后，最终可以在屏幕上显示。这样就可以预先剔除不可见的图元，仅对剩余的图元进行处理。这个过程也称为消隐。剔除的目的是将不会在屏幕上显示的图元预先排除，从而减少图形绘制的计算量。常见的剔除算法包括视域剔除、小物体剔除、背面剔除、退化剔除等。

1. 视域剔除

视域剔除算法是认为视域体以外的物体模型都是在屏幕之外，从而仅需对位于视域体之内的模型进行光栅化处理(如图2.9(a)所示)。在进行视域剔除时，需要判断场景中的模型和视域体是否发生了相交。然而，精确判断两个几何体的相交情况是非常困难的，尤其是当几何体的形状比较复杂时。为了便于图形流水线处理，通常利用包围盒进行模型相交的快速判断。包围盒是平行于物体空间坐标平面且包围整个物体的最小六面体，也就是能够容纳该物体的最小长方体。这样就可以将物体与视域体之间的求交问题，转化为物体包围盒与视域体的求交问题。显而易见，后者的计算量则大为降低。

2. 小物体剔除

小物体剔除算法是指小于特定尺寸的物体预先直接被剔除，不再进行任何光栅化处理。这类物体经过投影变换到屏幕上后，小于若干指定大小的像素区域，因此往往不需要进行绘制(如图2.9(b)所示)。在实际处理时，通常是计算物体的包围盒，然后剔除那些变换到屏幕后包围盒尺寸比一个像素阈值还小的物体模型。

3. 背面剔除

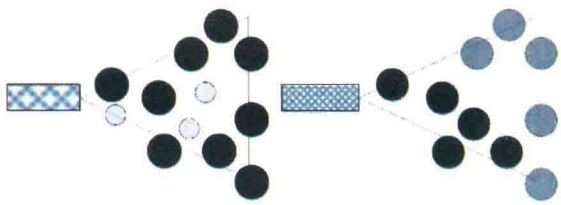
背面剔除算法是对于有正面和背面之分的封闭物体的剔除操作。当视点位置固定时，背对眼睛的面是不可见的，因此不需要进行光栅化处理。例如，对于三角面片来说，如果其正面是背对眼睛的，则它是不可见的。数学上，通过三角面片法向和投影线的夹角进

行判断是否为背面(如图 2.9(c)所示)。据统计,在由三角面片组成的物体模型中,大约50%的三角面片处于背面位置。因此,背面剔除能够减少绘制时一半的计算量。在背面剔除时,通常设定三角形的顶点索引值递增方向为逆时针方向,也就是可见方向;反之,则不可见。通过这个简单运算,就可以快速判断哪些面是需要绘制的正面,而哪些面是不需要绘制的背面,从而提高模型的整体绘制速度。

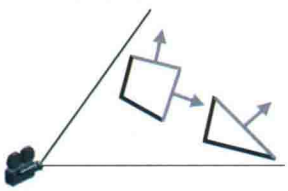
4. 退化剔除

退化剔除算法是对面积为零的几何图元不进行光栅化处理。这种面积为零的几何图元就称为退化图元。例如,图 2.9(d)所示的三角形的三个顶点位于同一条直线上,或者三个顶点位于同一个位置,或者三角面片的法向量为零等。在这些情况下,图元光栅化时可以直接忽略,从而进一步提高绘制效率。

(a) 视域剔除



(b) 小物体剔除



(c) 背面剔除

(d) 退化剔除

图 2.9 剔除方法

2.3.4 可见性判断

剔除操作主要面向场景中单个物体模型。如果场景中包含多个物体模型,则需要根据模型到眼睛距离的远近进行光栅化处理,也就是深度测试。这个过程又被称为图元的消隐或可见性判断。这类操作主要是根据前后模型之间的深度值差异,获取不同物体的相互遮挡关系,从而确定投影变换后覆盖的像素具体是要被哪一个模型所填充。常见的可见性判断算法有画家算法、深度缓存算法等。

画家算法也称为优先填充,是将场景中的图元根据深度进行排序,然后按照前后顺序进行绘制。这种算法在执行时会将不可见的部分覆盖,以此解决可见性问题。画家算法思想简单,是一种过时的算法。

目前的图形流水线广泛采用深度缓存算法,也就是 z-buffer 机制,在光栅化时进行高效的深度测试。事实上,z-buffer 是和绘制窗口尺寸相同的一块缓存,记录了每个填充后的像素和眼睛的距离,也就是 z 值。在开始绘制场景前,先把 z-buffer 中所有的值初始化

为无穷大，也就是物体位于无穷远处。然后，在绘制图元面片时，对面片的每个像素计算相应的 z 值，并和 z-buffer 中当前已存放的 z 值进行比较。如果 z-buffer 中的 z 值较大，就表示目前要填充的像素到眼睛的距离更近，所以应该在屏幕上绘制，并同时更新 z-buffer 中的 z 值。如果 z-buffer 中的 z 值较小，那就表示目前要填充的像素是比较远的，会被当前缓存中的像素遮挡，所以就不需要填充，也就不用更新 z 值。通过对深度 z 值的记录，就可以用任意的顺序来绘制这些三角面片，得到正确的绘制结果(如图 2.10 所示)。

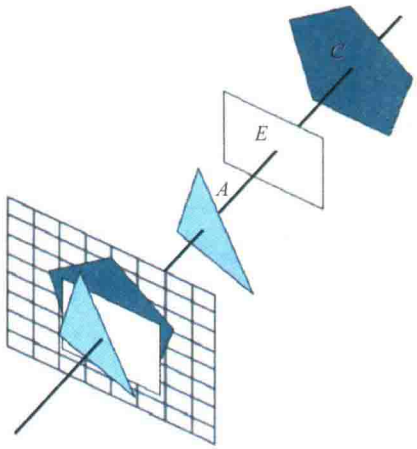


图 2.10 基于 z-buffer 的可见性判断

例题 2-6 z-buffer 算法的深度测试。

问题： 写出采用 z-buffer 深度测试进行可见性判断的伪代码。

解答： 给定多个模型 models，采用 z-buffer 深度测试对可见性进行判断的伪代码是

```
function Draw (ModelList models)
    z_buffer[screen_width][screen_height] ← max          /* 初始化最大深度值 */
    color_buffer[screen_width][screen_height] ← black  /* 初始化像素颜色值 */
    for each model in models do
        for each pixel in model.pixels do
            (x, y) ← pixel.position
            z ← get_depth (pixel)
            if z <z_buffer[x][y] then                          /* 深度判断 */
                z_buffer[x][y] ← z
                color_buffer[x][y] ← get_color (pixel)
            end if
        end for
    end forend function
```

□

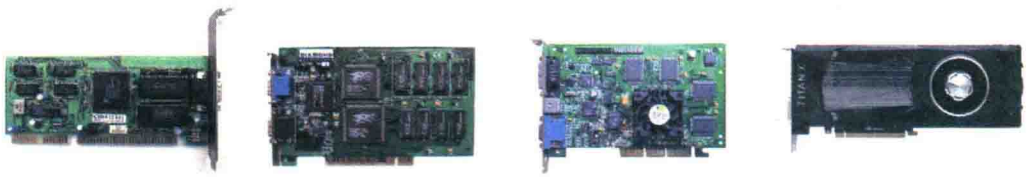
除了以上光栅化处理的步骤，实际图形流水线上还包含更多复杂的操作，例如纹理贴图、(半)透明、反走样、阴影、雾化、模糊等现象的处理，以获得更加多样和丰富的绘制效

果。通过在流水线上开闭或改变这些操作及其状态参数，就可以对场景中物体模型进行准确绘制与显示。

2.4 图形硬件

计算机的硬件组成中，很多部件与计算机图形学的建模、绘制和交互有着密切的关系。例如键盘和鼠标可以作为交互工具，显示器则是进行图形显示的设备等。在计算机硬件中，显卡是与计算机图形学最为紧密的设备，承担着输出显示图形的任务。可以说，显卡就是专为图形绘制而设计的，已成为计算机硬件组成不可缺少的部分。

早期的显卡是集成在主板上的一块芯片，主要用于文字数据的显示，不具备图形处理的能力。如图 2.11 所示，第一块独立的显卡出现在 20 世纪 90 年代初期，Trident 8900/9000 显卡是当时 2D 图形显卡的代表。这个时期的显卡主要用于加速图形流水线的若干步骤，将一些简单算法，如线绘制、多边形绘制、三角形填充等固化到显卡上专门的芯片内，这样可以适当加快图形绘制的速度。但图形流水线上更多的操作还是由 CPU 来完成。



Trident8900/9000显卡　Voodoo显卡　GeForce256显卡　GeForce3显卡

图 2.11　几款经典的显卡

1996 年，美国 3Dfx 公司推出的 Voodoo 显卡，虽然只有 4MB 显存、50MHz 频率的处理速度，但成为了图形硬件发展过程中的里程碑。其主要意义在于开始支持 3D 图形的绘制，并以专门的芯片设计提高绘制效率。此后，显卡也开始成为计算机不可或缺的重要硬件，但大量的图形绘制算法还是依赖于 CPU 的计算。

1999 年，美国 Nvidia 公司提出了 GPU（Graphics Processing Unit）的概念，并在 GeForce 256 显卡中实际应用。GPU，中文名为图形处理单元，是一种由数以千计的更小、更高效的核心组成的大规模并行计算架构。GPU 的出现，使得显卡减少了对 CPU 的依赖，尤其是在 3D 图形处理时取代了部分原本 CPU 的工作。接下来的十多年时间，GPU 的硬件水平和计算能力取得了飞速发展。

进入 21 世纪，支持可编程 GPU 的显卡开始出现。2001 年，Nvidia 公司推出的 GeForce 3 显卡，可以通过顶点着色器的编程进行几何操作。常用的编程语言如 C、C++ 等都有面向 GPU 的版本，从而使 GPU 的使用更加方便。2013 年，Nvidia 公司发布的 GeForce TITAN 显卡，包含有 70 亿个晶体管、6G 显存，因其强大的计算能力而广泛应用于高性能计算、无人驾驶等领域。

2.5 小 结

图元是构成图形的基本元素，经过图形流水线上的几何变换、光栅化处理等，最终绘制成由像素组成的二维图像，进而在显示器等终端进行呈现。整个图形流水线可以看作状态机，通过更改图元操作的属性值，实现指定效果的图形到图像的转化。

思考题

2.1 计算机图形流水线的主要步骤有哪些？它们分别负责什么样的图元处理？

2.2 计算机图形流水线上的几何变换有哪些？它们分别完成什么样的坐标系转换？

2.3 几何变换时为什么采用齐次坐标进行矩阵和向量运算？

2.4 透视投影和正视投影的数学表示形式和变换效果有什么不同？它们各自的用途是什么？

2.5 如何判断平面上的一个点在三角形的内部还是外部？

2.6 光栅化处理时的剔除算法有哪些？

2.7 深度测试的 z-buffer 算法原理是什么？

第3章 几何建模

计算机图形学的早期发展动力源自于CAD/CAE/CAM系统中对几何造型的需求。它们需要通过计算机表示、存储、分析、控制并输出指定形状的几何模型。这里的形状既可以是单个物体的外形，也可以是由一组物体所构成的复杂场景的外形。此外，在各种类型图形的真实感绘制过程中，模型的几何形状也会直接影响其表面光照分布、纹理密度等因素，进而与模型的绘制效果也是密切相关的。因此，长期以来几何建模都是计算机图形学中的重要内容。

本章介绍几何建模涉及的数学基础知识，包括形状表达的数学形式、常用几何性质等。接下来围绕三种重要的几何建模技术：自由曲线/曲面建模、细分曲面建模和三维重建，介绍相应的概念、模型及其使用方法。此外，还介绍若干其他类型的建模方法，例如基于分形、粒子系统、语言学等的方法。最后，针对计算机内部数据存储特点，介绍面向几何建模的典型数据结构。

3.1 数学基础

一般来讲，计算机图形学中的几何建模是指构造物体或场景二维/三维形状的数学表达形式及其在计算机内表示的数据结构。通常意义下，这种数学形式具有直接、明确的函数关系，使得所有满足该关系的形状也具有相应的几何性质。因此，首先需要了解几何形状的数学形式。

3.1.1 几何形状的数学形式

数学上，连续几何形状的表达方式主要有三种：显式表达式、隐式表达式和参数表达式。其中，显式表达式和隐式表达式又称为代数形式。这三种形式的主要区别在于定义函数时，其因变量和自变量的表达方式和相互之间代数制约关系的不同。下面分别介绍这三种表达式。

1. 显式表达式

显式表达式是一种因变量随自变量变化而变化的函数形式。二维平面上几何形状的显式表达式通常为 $y=f(x)$，其中 x 是自变量，y 是因变量。例如，$y=2x+1$ 代表平面直线，$y=x^2+x+1$ 代表平面抛物线。三维空间中几何形状的显式表达式通常为 $z=f(x,y)$，其中 x 和 y 是自变量，z 是因变量。例如，$z=2x+y+1$ 表示了三维空间中的一张平面，$z=\sqrt{1-x^2-y^2}$ 表示了半径为1的半球面。从表达形式上看，显式表达式的因变量和

自变量分别位于等号两侧，二者具有明显的对应关系。

2. 隐式表达式

隐式表达式是由多个变量共同定义的一种函数形式。这里的变量是没有自变量和因变量之分的，不能单独进行表示。二维平面上几何形状的隐式表达式通常具有 $f(x,y)=0$ 的形式。例如，$x^2+y^2-1=0$ 表示了平面圆形。三维空间中几何形状的隐式表达式具有 $f(x,y,z)=0$ 的形式。例如，$x^2+y^2+z^2-1=0$ 表示了三维球面。从表达形式上看，隐式表达式的因变量和自变量是混合在一起的，都位于等式的同侧，二者没有明显的对应关系。

3. 参数表达式

参数表达式是采用若干独立变量作为自变量的显式表达式组成的集合。这种表达方式下几何形状是由指定集合的参数作为自变量而得到的因变量所表示的。作为自变量的参数也称为参变量。例如，二维平面和三维空间中的几何形状分别具有如下表达形式：

$$\begin{cases} x = f(t) \\ y = g(t) \end{cases} \qquad \begin{cases} x = f(u,v) \\ y = g(u,v) \\ z = h(u,v) \end{cases} \tag{3.1}$$

其中，t、u 和 v 是参变量。它们在给定取值范围内变化，从而产生相应的几何形状。

从代数形式来讲，参数表达式也可以看作为一种显式表达式。但是，参数表达式具有更加直观的几何意义，反映了参变量变化所产生的各个维度的变化情况。例如，曲线可以看作单个参变量在指定区间内变化所形成的空间点的集合；而曲面则可以看作双参变量变化形成的空间点的集合。此外，参数表达式的代数和微积分运算非常方便，适合对几何形状的性质进行分析。

对于一些几何形状，可能存在多种不同形式的表达式。如图 3.1 所示，三维空间中半径为 1 的球面，除了上述隐式表达式，也可以采用基于球极坐标的参数表达式，也就是

$$\begin{cases} x = \sin\varphi\cos\theta \\ y = \sin\varphi\sin\theta \\ z = \cos\varphi \end{cases} \tag{3.2}$$

其中，参变量 θ 和 φ 分别对应于方位角和俯仰角。但是对于球面而言，一般不存在直接的显式表达式。

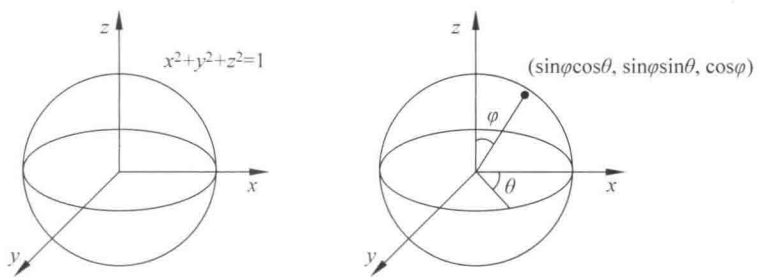
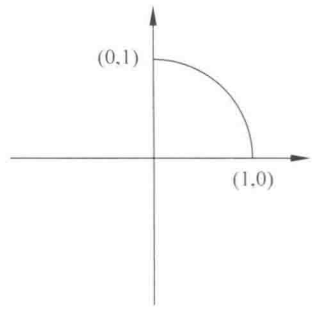


图 3.1 球面的隐式表达式与参数表达式

例题 3-1　平面曲线的数学表达式。

问题：如右图所示，平面坐标系第一象限内的单位圆弧的显式表达式和参数表达式分别是什么？



解答：该圆弧的显式表达式可以写为：

$$y = \sqrt{1-x^2}, \quad 0 \leqslant x \leqslant 1$$

对应的参数表达式可以写为：

$$\begin{cases} x(t) = (1-t^2)/(1+t^2) \\ y(t) = (2t)/(1+t^2) \end{cases}, \quad 0 \leqslant t \leqslant 1$$

□

3.1.2　几何性质

通过几何形状的数学形式，可以研究模型所具备的几何性质，从而更好地设计和控制建模效果。计算机图形学中几何建模所涉及的形状主要包括两种类型：曲线和曲面。曲线又包括二维平面曲线和三维空间曲线，而曲面主要是指三维空间中的曲面。这些也是在日常生产生活中，人们接触最多的物体形状。

1. 二维平面曲线

二维平面曲线理论上可以采用显式、隐式或参数形式进行表达。显式形式是具有一个自变量和一个因变量的表达式，也就是 $y=f(x)$。隐式形式是由两个变量构成的方程式，也就是 $f(x,y)=0$。而参数形式则是单参变量表示的二维向量，也就是 $\gamma(t)=(x(t),y(t))$。例如，平面上椭圆的隐式表达式和参数表达式分别具有如下形式：

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1, \quad \begin{cases} x = a\cos\theta \\ y = b\sin\theta \end{cases} \tag{3.3}$$

其中，a 和 b 是非零常数，θ 是参变量。平面曲线常用的几何性质有曲线弧长、曲率等，反映了曲线本身的形状。

以参数形式表达的曲线为例，它的弧长描述了曲线在某一参变量取值区间内的测度，计算公式为

$$l = \int_{t_0}^{t_1} \sqrt{\dot{x}^2(t)+\dot{y}^2(t)}\,\mathrm{d}t$$

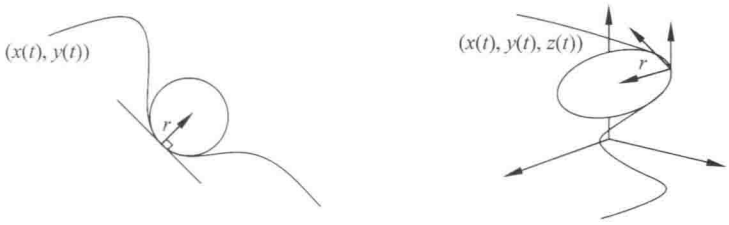
其中，$\dot{x}(t)$ 和 $\dot{y}(t)$ 是函数关于参变量 t 的一阶导数，$[t_0,t_1]$ 是 t 的取值区间。曲率定义为切线方向角相对于弧长的变化率，计算公式为

$$\kappa(t) = \frac{|\dot{x}(t)\ddot{y}(t)-\ddot{x}(t)\dot{y}(t)|}{(\dot{x}^2(t)+\dot{y}^2(t))^{\frac{3}{2}}}$$

其中，$\ddot{x}(t)$ 和 $\ddot{y}(t)$ 是函数关于参变量 t 的二阶导数。实际上，曲率反映了曲线在某一点的弯曲程度，例如直线的曲率处处为 0。此外，平面曲线上某一点的曲率大小等于该点处密切圆半径 r 的倒数，而密切圆的圆心位于曲线凹向的一侧（如图 3.2(a)所示）。

2. 三维空间曲线

三维空间曲线可以采用单个参变量作为参数的三维向量进行表示，也就是 $\gamma(t)=(x(t),y(t),z(t))$。它直观上反映了质点在三维空间中单自由度的运动轨迹。这里需要



(a) (b)

图 3.2 平面二维曲线和三维空间曲线的曲率圆

注意的是，三维空间曲线一般不存在显式表达式，而其隐式表达式由两张曲面的隐式表达式组成的方程组所构成，可以记为

$$\begin{cases} f(x,y,z) = 0 \\ g(x,y,z) = 0 \end{cases}$$

它反映了三维空间曲面可以看作两张曲面相交而产生。弧长、曲率和挠率是空间曲线的三种基本属性，反映了三维曲线的形状。与平面曲线类似，弧长表示了指定区间内的曲线长度，计算公式为

$$l = \int_{t_0}^{t_1} \sqrt{\dot{x}^2(t) + \dot{y}^2(t) + \dot{z}^2(t)}\,\mathrm{d}t$$

曲率反映了曲线在空间中的弯曲程度，计算公式为

$$\kappa(t) = \frac{|\,\dot{\gamma}(t) \times \ddot{\gamma}(t)\,|}{(|\,\dot{\gamma}(t)\,|)^{\frac{3}{2}}}$$

其中，$\dot{\gamma}(t)$ 和 $\ddot{\gamma}(t)$ 分别是曲线所对应三维向量的一阶和二阶导数。如图 3.2(b)所示，曲率也与该点密切圆半径 r 成反比关系。与平面曲线不同，空间曲线具有挠率属性，计算公式为

$$\tau(t) = \frac{(\dot{\gamma}(t) \times \ddot{\gamma}(t)) \cdot \dddot{\gamma}(t)}{|\,\dot{\gamma}(t) \times \ddot{\gamma}(t)\,|^2}$$

挠率反映了曲线切平面的扭转状况。例如，所有平面曲线的挠率都为 0。

3. 三维空间曲面

三维空间曲面理论上可以采用显式、隐式或参数形式进行表达。显式形式是具有两个自变量和一个因变量的表达式，通常记为 $z=f(x,y)$。隐式形式是由三个变量构成的方程式，也就是 $f(x,y,z)=0$。参数形式则是由两个参变量表示的三维向量形式，通常记为 $\pi(u,v)=(x(u,v),y(u,v),z(u,v))$。例如，图 3.3 所示环面的隐式表达式和参数表达式分别为：

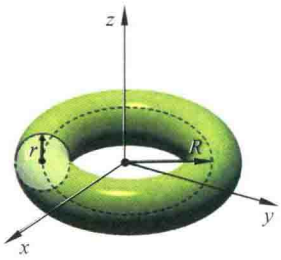


图 3.3 环面

$$(x^2+y^2+z^2+R^2-r^2)^2 - 4R^2(x^2+y^2) = 0, \quad \begin{cases} x(u,v) = (R+r\cos v)\cos u \\ y(u,v) = (R+r\cos v)\sin u \\ z(u,v) = r\sin v \end{cases} \tag{3.4}$$

其中 R 和 r 分别表示环截面半径和内环半径。

空间曲面常用的几何性质涉及面积、法曲率、主曲率、高斯曲率、平均曲率等属性。以参数形式表示的曲面为例，其面积是指双参变量在取值范围内的表面测度，计算公式为

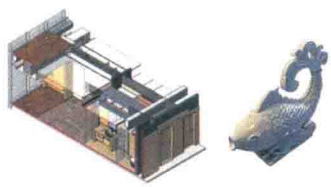
$$s = \int_{u_0}^{u_1}\int_{v_0}^{v_1} | \dot{\pi}_u \times \dot{\pi}_v | \, \mathrm{d}u\mathrm{d}v$$

其中$[u_0, u_1]$和$[v_0, v_1]$分别是参变量 u 和 v 的取值范围。曲面上某点处的曲率和经过该点的曲面上曲线的弯曲程度相关，称为法曲率。事实上，曲面在一点处有无穷多个切方向，因此可以定义无穷多个法曲率。其中，法曲率的最大值和最小值称为主曲率，代表了该点处法曲率的极值分布情况，也就是能达到的最大和最小弯曲程度。曲面上某点处主曲率的平均值，称为平均曲率，而主曲率的乘积则称为高斯曲率。事实上，平均曲率和高斯曲率是反映曲面局部形状的两个重要属性。

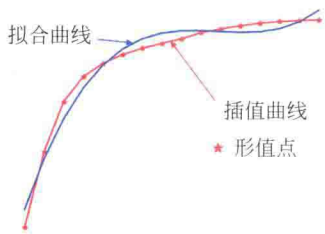
3.1.3 建模工具

根据建模对象的不同，几何建模可以简单地分为自然物体建模和人造物体建模。所谓自然物体建模，是指建模对象来自于自然界中正常存在的物体，例如各种动物、植物、地形、地貌等(如图 3.4(a)所示)。这类物体的几何形状往往很难用精确的数学公式直接表示。人造物体建模，是指通过手工交互或自动化的方式，利用相应的工具对基本几何形状进行变换和重组，获得具有新形状的物体(如图 3.4(b)所示)。这类物体的几何形状，通常可以借助变换或重组时的数学表达式进行准确描述。

(a) 自然物体建模



(b) 人造物体建模



(c) 型值点的插值与拟合

图 3.4 物体对象几何建模的方式：插值与拟合

无论自然物体建模，还是人造物体建模，主要有插值和拟合两种方式(如图 3.4(c)所示)。这两种方式都是先对物体形状有个大概的主观描述，在此基础上按照精度的要求进

行模型的构建。插值，是要求模型能够严格满足给定数据点的几何位置约束。这些数据点也称为型值点，是对建模对象外形的离散采样。拟合则是在一定的几何意义下，建模的形状尽可能逼近给定的型值点。不论对模型进行插值或拟合，常用的数学工具有自由曲线/曲面、细分曲面等，它们为自然物体建模和人造物体建模提供了有效手段。

3.2　自由曲线/曲面建模

经典几何形状，如平面、圆柱面、圆锥面等都具有明确的解析表达式。然而，绝大部分的自然物体，例如人脸等，其形状往往无法使用一个解析函数进行显式表示，需要构造新的函数形式来表示其形状。所谓自由曲线/曲面，是指不能用初等解析函数完全准确地表达全部形状的曲线和曲面。因此，自由曲线/曲面的出现，是为了克服使用解析函数进行形状表示的局限性，采用更广泛意义下的数学函数来表示形状。

3.2.1　平面三次多项式曲线

多项式曲线是一种最常见的平面曲线表达形式。在给定型值点集合后，计算 n 次多项式进行插值或拟合。这样就可以转化为线性方程组来求解该多项式。数学上，多项式次数越高，其对应曲线出现的拐点就越多，能表示的形状也就越复杂。但是次数越高，越可能在拟合时出现过拟合，难以控制那些没有采样点处的形状。此外，五次以上多项式的计算比较复杂，不利于快速建模。如果多项式次数较低，又会出现欠拟合情况，导致形状的表现力不足。在实际应用中，三次多项式曲线是使用较广泛的一类多项式曲线。它对应的参数表达式为：

$$\boldsymbol{p}(t) = \boldsymbol{c}_0 + \boldsymbol{c}_1 t + \boldsymbol{c}_2 t^2 + \boldsymbol{c}_3 t^3 \tag{3.5}$$

其中，$\boldsymbol{c}_i = (x_i, y_i)^T$ 称为控制顶点，$t \in [0,1]$ 是参变量取值范围。通过公式(3.5)可以看出，$\{\boldsymbol{c}_i\}$ 取值不同，就会产生不同的曲线形状。

如图 3.5 所示，给定 4 个型值点 $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$，可以计算通过这些型值点的一条三次多项式曲线。这是由于三次多项式总共有 4 个系数，每个系数同时有 x 和 y 两个分量，因此一共有 8 个未知数。而 4 个型值点恰好可以提供 8 个约束条件。如果给定的型值点多于 4 个，那么可以通过最小二乘法计算一条最优的三次多项式曲线来拟合这些型值点，使得它们距离曲线的总体几何距离最小。

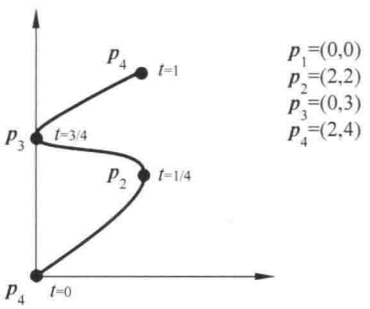


图 3.5　由 4 个型值点定义的三次多项式曲线

例题 3-2　平面三次多项式插值曲线。

问题： 计算图 3.5 所示的多项式曲线的参数表达式 $\boldsymbol{p}(t)$。

解答： 根据 4 个型值点 $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$ 的 x 和 y 坐标，分别计算 $\boldsymbol{p}(t) = (x(t), y(t))$ 控制顶点 $\boldsymbol{c}_i = (x_i, y_i)^T$。其中，$x$ 和 y 坐标分别满足如下等式：

$$\begin{cases} 0 = x_0 \\ 2 = x_0 + \frac{1}{4}x_1 + \frac{1}{16}x_2 + \frac{1}{64}x_3 \\ 0 = x_0 + \frac{3}{4}x_1 + \frac{9}{16}x_2 + \frac{27}{64}x_3 \\ 2 = x_0 + x_1 + x_2 + x_3 \end{cases}$$

$$\begin{cases} 0 = y_0 \\ 2 = y_0 + \frac{1}{4}y_1 + \frac{1}{16}y_2 + \frac{1}{64}y_3 \\ 3 = y_0 + \frac{3}{4}y_1 + \frac{9}{16}y_2 + \frac{27}{64}y_3 \\ 4 = y_0 + y_1 + y_2 + y_3 \end{cases}$$

通过求解上述两个线性方程组，可以得到 4 个控制顶点的坐标分别是 $\boldsymbol{c}_0=(0,0)$、$\boldsymbol{c}_1=(18,12)$、$\boldsymbol{c}_2=(-48,-18.67)$和 $\boldsymbol{c}_3=(32,10.67)$。

□

平面三次多项式能够通过插值或拟合的方式对平面曲线建模，但是像公式(3.5)这样的表达式缺少直观的几何解释(例如，其控制顶点无法直接描述曲线形状)，这样就不利于工程人员按照主观意图进行曲线建模来设计相应的形状。此外，受单个多项式代数性质的影响，能有效表示的曲线形状有限，并不适合更加丰富多样的几何外形设计。因此，需要更灵活的自由曲线/曲面建模技术，典型的建模技术有 Bézier 曲线/曲面和 B 样条曲线/曲面。

3.2.2 Bézier 曲线/曲面

Bézier 曲线是以法国工程师 Pierre Bézier 命名的。而 Bézier 曲线的计算方法最早可以追溯到法国雷诺公司工程师 Paul de Casteljau。他提出了适用于计算机编程的递归算法，只是没有意识到这种算法最后所生成的曲线形状就是 Bézier 曲线的形状。1962 年，Bézier 明确给出了这种曲线的数学公式，并成功地应用于汽车外形设计。

Bézier 曲线是第一种在工业界被广泛采用的自由曲线建模工具，其本质也是多项式曲线。但是，这种曲线具有更直观的几何表达形式，能够方便工程师通过操纵控制顶点来进行建模。具体地讲，n 次 Bézier 曲线是一种单参变量的参数表达式，数学上定义为：

$$\boldsymbol{p}(t) = \sum_{i=0}^{n} B_i^n(t)\boldsymbol{c}_i = \sum_{i=0}^{n} \binom{n}{i}(1-t)^{n-i}t^i\boldsymbol{c}_i, \quad t \in [0,1] \tag{3.6}$$

其中，$\{\boldsymbol{c}_0,\boldsymbol{c}_1,\cdots,\boldsymbol{c}_n\}$是 $n+1$ 个控制顶点。这些控制顶点依次连接形成控制多边形。单参变量函数 $B_i^n(t)$也称为 Bernstein 多项式函数。图 3.6 分别展示了二次和三次 Bézier 曲线。

数学上，多项式函数空间是可以采用 Bernstein 函数作为一组基函数。因此，Bézier 曲线就是采用这组基函数的线性组合来表示的一类多项式函数。在此基础上，可以将公式(3.5)所表示的任意形式的多项式曲线转化为公式(3.6)表示的 Bézier 曲线。这个过程实质上就是将多项式基函数$\{1,t,t^2,t^3\}$转化为 Bernstein 基函数表示。

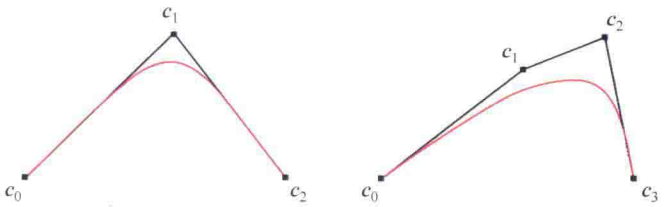


图 3.6 二次和三次 Bézier 曲线

Bézier 曲线之所以在工业设计中广受欢迎，主要是因为公式(3.6)定义的几何形状具有以下良好性质，能够方便设计人员对形状进行控制。

(1) 首末端点插值。曲线经过控制多边形的首末端点，也就是曲线端点满足 $\boldsymbol{p}(0)=\boldsymbol{c}_0$ 和 $\boldsymbol{p}(1)=\boldsymbol{c}_n$。此外，曲线在两个端点处的切线方向与控制多边形的边平行，也就是曲线插值首末端点的切向量。

(2) 保凸性。曲线位于控制多边形构成的凸包(凸多边形边界)内。这样在给定控制多边形后，就可以通过凸包来限定生成的 Bézier 曲线的范围。这个性质方便设计人员通过控制顶点对形状进行调控。

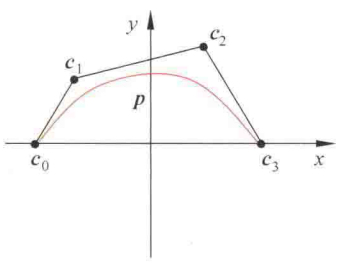
(3) de Casteljau 递归算法。在计算公式(3.6)表示的 Bézier 曲线时，除了直接采用多项式的代数运算，还可以借助 de Casteljau 递归算法实现更加快速地计算。该递归算法基于以下递推公式：

$$\begin{cases}\boldsymbol{p}_i^{[r]}(t)=(1-t)\cdot\boldsymbol{p}_{i-1}^{[r-1]}(t)+t\cdot\boldsymbol{p}_i^{[r-1]}(t)\\ \boldsymbol{p}_i^{[0]}(t)\equiv\boldsymbol{p}_i^{[0]}=\boldsymbol{c}_i\end{cases}\tag{3.7}$$

上述 de Casteljau 递归算法证明了当 $r=n$ 时，$\boldsymbol{p}_n^{[n]}(t)$一定是位于 Bézier 曲线上的点。这种递归算法非常适合计算机编程实现，计算效率很高。

例题 3-3 平面三次 Bézier 曲线。

问题：写出右图所示的控制顶点定义的三次 Bézier 曲线，其中 4 个控制顶点坐标分别是 $\boldsymbol{c}_0(-2,0)$，$\boldsymbol{c}_1(-1,1)$，$\boldsymbol{c}_2(1,1.5)$，$\boldsymbol{c}_3(2,0)$。然后，给出采用 de Casteljau 递归算法计算曲线上任意一点的递归过程。



解答：将 4 个控制顶点的坐标代入公式(3.6)，可以得到如下相应的 Bézier 曲线表达式：

$$\begin{aligned}p(t)&=\sum_{i=0}^{3}B_i^3(t)\boldsymbol{c}_i\\&=\begin{pmatrix}-2B_0^3(t)-B_1^3(t)+B_2^3(t)+2B_3^3(t)\\B_1^3(t)+1.5B_2^3(t)\end{pmatrix}\\&=\begin{pmatrix}-2(1-t)^3-3(1-t)^2t+3(1-t)t^2+2t^3\\3(1-t)^2t+4.5(1-t)t^2\end{pmatrix}\end{aligned}$$

其中，$t\in[0,1]$。进一步，根据公式(3.7)定义的 de Casteljau 递归算法，对于曲线上任意一点 $\boldsymbol{p}(t_0)$，$0\leqslant t_0\leqslant 1$，其递归计算过程如下：

$$\boldsymbol{p}(t_0)=\boldsymbol{p}_3^{[3]}(t_0)=(1-t_0)\boldsymbol{p}_2^{[2]}(t_0)+t_0\boldsymbol{p}_3^{[2]}(t_0)$$

$$\begin{cases}\boldsymbol{p}_2^{[2]}(t_0)=(1-t_0)\boldsymbol{p}_1^{[1]}(t_0)+t_0\boldsymbol{p}_2^{[1]}(t_0)\\ \boldsymbol{p}_3^{[2]}(t_0)=(1-t_0)\boldsymbol{p}_2^{[1]}(t_0)+t_0\boldsymbol{p}_3^{[1]}(t_0)\end{cases}$$

$$\begin{cases}\boldsymbol{p}_1^{[1]}(t_0)=(1-t_0)\boldsymbol{p}_0^{[0]}(t_0)+t_0\boldsymbol{p}_1^{[0]}(t_0)\\ \boldsymbol{p}_2^{[1]}(t_0)=(1-t_0)\boldsymbol{p}_1^{[0]}(t_0)+t_0\boldsymbol{p}_2^{[0]}(t_0)\\ \boldsymbol{p}_3^{[1]}(t_0)=(1-t_0)\boldsymbol{p}_2^{[0]}(t_0)+t_0\boldsymbol{p}_3^{[0]}(t_0)\end{cases}$$

其中，$\boldsymbol{p}_0^{[0]}(t_0)=\boldsymbol{c}_0$，$\boldsymbol{p}_1^{[0]}(t_0)=\boldsymbol{c}_1$，$\boldsymbol{p}_2^{[0]}(t_0)=\boldsymbol{c}_2$ 以及 $\boldsymbol{p}_3^{[0]}(t_0)=\boldsymbol{c}_3$。

□

进一步，Bézier 曲面是由两个参变量的 Bernstein 混合函数表示的参数曲面。具体来讲，$m\times n$ 次 Bézier 曲面是由$(m+1)\times(n+1)$个控制顶点组成的多面体(也称为控制网格)来定义，记为：

$$\boldsymbol{p}(s,t)=\sum_{i=0}^{m}\sum_{j=0}^{n}\boldsymbol{c}_{ij}B_i^m(s)B_j^n(t),\quad s\in[0,1],\ t\in[0,1] \tag{3.8}$$

与 Bézier 曲线类似，Bézier 曲面也具有很多良好的性质。例如，Bézier 曲面的边界是由 4 个边界多边形作为控制多边形所定义的 4 条 Bézier 曲线；角点插值性，也就是说 Bézier 曲面在 4 个角点插值控制网格的顶点；凸包性，也就是说 Bézier 曲面位于曲面控制网格形成的凸包内。

与一般的平面多项式曲线/曲面表达式相比，Bézier 曲线/曲面的主要优点是容易编程实现、具有端点和切向插值等性质，而且方便各种微积分运算的数值求解。但 Bézier 曲线/曲面的缺点也很明显，就是缺乏对曲线形状的局部可控性。具体来讲，改变其中一个控制顶点的位置，就会改变整个曲线/曲面的形状。这种局限性也限制了 Bézier 曲线/曲面的几何建模能力。

3.2.3 B 样条曲线/曲面

为了克服 Bézier 曲线/曲面的不足，美国 Utah 大学的 William Gordon 和 Richard Riesenfeld 在 1974 年将 B 样条参数曲线引入几何建模。样条，源于生产实践，本意是指富有弹性的细长条。样条利用压铁使其通过指定的型值点，并调整样条使它具有满意的形状，然后沿样条画出曲线。B 样条曲线则是通过 B 样条函数表示的曲线，本质上是分段多项式曲线。因此，B 样条曲线是由多个在连接处满足一定连续性的多项式曲线所构成的曲线，是一类多项式组合的曲线。在此之前，美国 Wisconsin-Madison 大学的 Isaac Schoenberg 早在 1946 年就利用 B 样条进行统计数据的光滑处理，开创了样条逼近的现代理论。

由于分段多项式的性质，B 样条曲线的定义除了需要控制顶点$\{\boldsymbol{c}_0,\boldsymbol{c}_1,\cdots,\boldsymbol{c}_n\}$，还需要设置节点向量 $t_0\leqslant t_1\leqslant\cdots\leqslant t_{n+k+1}$。在此基础上，$k$ 次($k+1$ 阶)B 样条曲线定义为：

$$\boldsymbol{p}(t)=\sum_{i=0}^{n}N_i^k(t)\boldsymbol{c}_i,\quad n\geqslant k \tag{3.9}$$

这里，相邻两个节点形成的区间$[t_k,t_{k+1})$定义了在其每一段上的多项式函数，而这些多项式函数的组合就定义了 B 样条曲线。图 3.7 展示了一条三次 B 样条曲线。

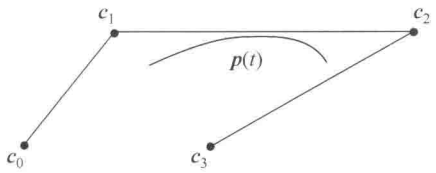


图 3.7 三次 B 样条曲线

具体来讲，在公式(3.9)中，$N_i^k(t)$是 B 样条基函数，可以通过如下递推公式来定义：

$$\begin{cases} N_i^0(t) = \begin{cases} 1, & t \in [t_i, t_{i+1}) \\ 0, & t \notin [t_i, t_{i+1}) \end{cases} \\ N_i^k(t) = \dfrac{t - t_i}{t_{i+k} - t_i} N_i^{k-1}(t) + \dfrac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(t) \end{cases} \tag{3.10}$$

公式(3.10)称为 de Boor-Cox 公式。由公式(3.10)可以看出，基函数 $N_i^k(t)$只有在位于节点 t_i 和 t_{i+k+1}之间的区间内取非负值，而在其他处则取值为零。这也意味着在区间 $[t_i, t_{i+1})$上，总共只有 $k+1$ 个 B 样条基函数取非负值，它们依次是 $N_{i-k}^k(t)$，$N_{i-k+1}^k(t)$，…，$N_i^k(t)$。基于该递归公式，B 样条曲线就可以采用类似 Bézier 曲线的递归方式进行计算，也就是通过逐次迭代的方式计算公式(3.8)表示的 B 样条曲线上任意点的位置坐标。这种计算方式称为 de Boor 递归算法。

该递归算法与 Bézier 曲线的 de Casteljau 递归算法类似，都是从控制顶点组成的控制多边形 $\boldsymbol{c}_{j-k}\boldsymbol{c}_{j-k+1}\cdots\boldsymbol{c}_j$ 开始，依次执行 k 次割角操作。其中，第 r 次割角是用线段 $\boldsymbol{p}_i^{[r]}(t)\boldsymbol{p}_{i+1}^{[r]}(t)$割去角 $\boldsymbol{p}_i^{[r-1]}$。这里 $\boldsymbol{p}_i^{[r]}(t)=(1-\lambda_{i,k-r+1})\boldsymbol{p}_{i-1}^{[r-1]}(t)+\lambda_{i,k-r+1}\boldsymbol{p}_i^{[r-1]}(t)$，而割角时线段端点在控制多边形边上的比例是 $\lambda_{i,k-r+1}=(t-t_i)/(t_{i+k-r+1}-t_i)$。那么，最后得到的角点 $\boldsymbol{p}_j^{[k]}(t)$就是 B 样条曲线上的点 $\boldsymbol{p}(t)$。整个递归割角过程可以用如下公式表示：

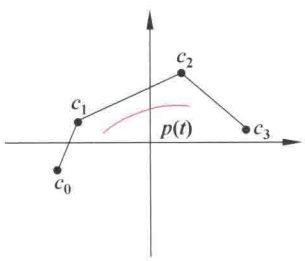
$$\boldsymbol{p}(t) = \sum_{i=j-k}^{j} N_i^k(t)\boldsymbol{p}_i^{[0]} = \sum_{i=j-k+1}^{j} N_i^{k-1}(t)\boldsymbol{p}_i^{[1]} \cdots = \sum_{i=j-1}^{j} N_i^1(t)\boldsymbol{p}_i^{[k-1]} \cdots = \boldsymbol{p}_j^{[k]} \tag{3.11}$$

其中，$\boldsymbol{p}_i^{[0]}=\boldsymbol{c}_i$ 是控制顶点。事实上，该 B 样条曲线是定义在区间$[t_j, t_{j+1})$上的一条多项式曲线。该区间也是公式(3.11)中能够使得所有 B 样条基函数取非零值的区间。

B 样条曲线保留了 Bézier 曲线良好的几何性质，同时具备了对曲线形状的局部可控性。具体来讲，根据 B 样条基函数的局部支撑性，改动其中一个控制顶点，B 样条曲线上仅仅和该控制顶点相关的曲线形状发生变化。此外，在进行多个 B 样条曲线拼接时，也可以根据节点设置很容易地保持拼接时的几何连续性。因此，B 样条曲线具有更加灵活的几何建模能力。

例题 3-4 平面三次 B 样条曲线。

问题：如右图所示的三次均匀 B 样条曲线 $\boldsymbol{p}(t)$的控制顶点分别是 $\boldsymbol{c}_0$、$\boldsymbol{c}_1$、$\boldsymbol{c}_2$、$\boldsymbol{c}_3$，节点间距相等，设为 $t_i=i$。写出 $t\in[t_3, t_4)$区间上的 B 样条曲线表达式。然后，给出采用 de Boor-Cox 递归算法计算曲线上任意一点的递归过程。



解答：根据公式(3.10)中 B 样条基函数的定义，可以得到区间$[t_3, t_4)$上 4 个非零基函数的表达式：

$$N_0^3(t)=(-t^3+3t^2-3t+1)/6, N_1^3(t)=(3t^3-6t^2+4)/6$$
$$N_2^3(t)=(-3t^3+3t^2+3t+1)/6, N_3^3(t)=t^3/6$$

因此，对应的三次均匀 B 样条曲线可以写为：

$$\boldsymbol{p}(t)=\sum_{i=0}^{3}N_i^3(t)\boldsymbol{c}_i=\frac{1}{6}(t^3,t^2,t^1,1)\boldsymbol{M}_{4\times4}\cdot\begin{bmatrix}\boldsymbol{c}_0\\ \boldsymbol{c}_1\\ \boldsymbol{c}_2\\ \boldsymbol{c}_3\end{bmatrix}$$

$$\boldsymbol{M}_{4\times4}=\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}$$

对于曲线上任意一点 $\boldsymbol{p}(t_0)$，$3\leqslant t_0<4$，其递归计算过程如下：

$$\boldsymbol{p}(t_0)=\boldsymbol{p}_3^{[3]}(t_0)=(4-t_0)\boldsymbol{p}_2^{[2]}(t_0)+(t_0-3)\boldsymbol{p}_3^{[2]}(t_0)$$

$$\begin{cases}\boldsymbol{p}_2^{[2]}(t_0)=\dfrac{4-t_0}{2}\boldsymbol{p}_1^{[1]}(t_0)+\dfrac{t_0-2}{2}\boldsymbol{p}_2^{[1]}(t_0)\\ \boldsymbol{p}_3^{[2]}(t_0)=\dfrac{5-t_0}{2}\boldsymbol{p}_2^{[1]}(t_0)+\dfrac{t_0-3}{2}\boldsymbol{p}_3^{[1]}(t_0)\end{cases}$$

$$\begin{cases}\boldsymbol{p}_1^{[1]}(t_0)=\dfrac{4-t_0}{3}\boldsymbol{p}_0^{[0]}(t_0)+\dfrac{t_0-1}{3}\boldsymbol{p}_1^{[0]}(t_0)\\ \boldsymbol{p}_2^{[1]}(t_0)=\dfrac{5-t_0}{3}\boldsymbol{p}_1^{[0]}(t_0)+\dfrac{t_0-2}{3}\boldsymbol{p}_2^{[0]}(t_0)\\ \boldsymbol{p}_3^{[1]}(t_0)=\dfrac{6-t_0}{3}\boldsymbol{p}_2^{[0]}(t_0)+\dfrac{t_0-3}{3}\boldsymbol{p}_3^{[0]}(t_0)\end{cases}$$

其中，$\boldsymbol{p}_0^{[0]}(t_0)=\boldsymbol{c}_0$，$\boldsymbol{p}_1^{[0]}(t_0)=\boldsymbol{c}_1$，$\boldsymbol{p}_2^{[0]}(t_0)=\boldsymbol{c}_2$ 以及 $\boldsymbol{p}_3^{[0]}(t_0)=\boldsymbol{c}_3$。

□

进一步，B 样条曲面是一种双参变量 B 样条函数组成的混合多项式曲面。B 样条曲面可以看作 B 样条曲线在三维空间沿另外一条 B 样条曲线滑动形成。例如双三次 B 样条曲面的表达式为 $\boldsymbol{p}(s,t)=\sum_{i=0}^{3}\sum_{j=0}^{3}N_i^k(s)N_j^k(t)\boldsymbol{c}_{ij}$。与 B 样条曲线类似，B 样条曲面也具有很多优良性质，例如局部性，即曲面形状只和最相关的几个控制顶点有关；凸包性，即 B 样条曲面的每一片都位于定义该片曲面的控制顶点的凸包之内；磨光性，即同一组控制顶点定义的 B 样条曲面，随着次数的升高越来越光滑。此外，Bézier 曲面也可以看作 B 样条曲面的特例。

然而，B 样条曲面无法直接表示圆锥等有理曲面。为此，研究人员又提出了非均匀有理 B 样条曲面(NURBS)。与普通的 B 样条曲线/曲面相比，NURBS 的特点是采用非均匀节点，同时是一种有理表示形式。与此同时，加入了权重因子，进一步控制曲线/曲面形状(如图 3.8 所示)。具体来讲，NURBS 具有如下表达式：

$$\boldsymbol{p}(s,t)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\boldsymbol{c}_{ij}w_{ij}N_i^k(s)N_j^k(t)}{\sum_{i=0}^{n}\sum_{j=0}^{m}w_{ij}N_i^k(s)N_j^k(t)} \tag{3.12}$$

其中 c_{ij} 是控制顶点，w_{ij} 是权重因子。

图 3.8 NURBS 定义的曲面形状

NURBS 兼具 B 样条曲线/曲面形状局部可调以及连续阶数可调的优点，又能像有理 Bézier 曲线可精确地表示圆锥曲线的特性。1991 年，国际标准化组织(ISO)在其正式发布的工业产品数据交换 STEP 标准中，把 NURBS 作为自由曲线/曲面的唯一定义，成为设计工业产品几何形状的唯一数学方法。许多国际著名的 CAD 软件也把 NURBS 作为几何造型工具的首选，例如 AutoCAD、CATIA 等软件。

3.3 细分曲面建模

B 样条曲线/曲面、NURBS 等自由曲线/曲面在 CAD 中已得到广泛的应用。然而，这类自由曲线/曲面对它们自身的控制多边形或控制网格的拓扑结构有严格要求，通常来说，只能定义在矩形网格上。然而，现实中几何建模对象的拓扑结构往往是复杂多样的，例如计算机动画中要表示人的头和人的手。此外，因为模型是活动的，要在曲面的连接处保持光滑也是需要解决的问题。

针对上述问题，科研人员进一步发明了细分曲面。所谓细分曲面，是指多面体按照指定的细分规则进行无穷细化的极限。细分曲面可以看作自由曲线曲面在任意拓扑定义域上的推广。例如，B 样条曲线的递归算法实际上就是对其控制多边形或控制网格的切割磨光过程。因此，人们自然地希望将这一算法推广到任意拓扑的多面体上去，也就是通过几何和拓扑的修改，重新添加边、顶点、面来重定义新的控制网格，以及移动顶点的空间位置来平滑该控制网格，并由此定义细分建模过程。这个过程所产生的极限曲面就是细分曲面。

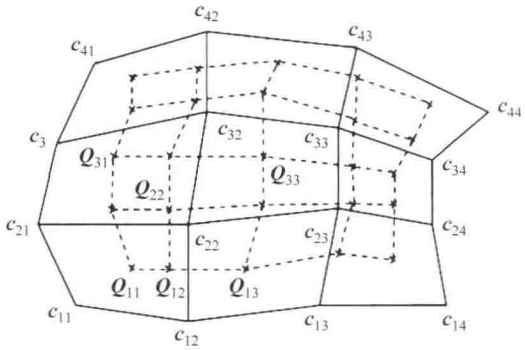
细分曲面提供了更加灵活、更好光滑性的曲面生成方法，早期主要应用于计算机动画领域。最典型的是 Pixar 公司的 Tony DeRose 将细分曲面应用于电影动画 *Geri's Game* 的制作，并获得了 1998 年奥斯卡最佳动画短片奖。对于细分曲面，最核心的是如何设计细分规则，也就是顶点、边、面的几何和拓扑生成方式。接下来主要介绍 4 种典型的细分规则及其生成的极限曲面：Catmull-Clark、Doo-Sabin、Loop 和 Butterfly 细分曲面。

3.3.1 Catmull-Clark 细分曲面

Catmull-Clark 细分曲面将双三次 B 样条曲面的生成方法推广到任意拓扑的控制网格，由 Edwin Catmull 和 Jim Clark 于 1978 年首次提出。普通的双三次 B 样条曲面由 16 个控制顶点 $\{c_{ij}\}_{i=1,j=1}^{4,4}$ 定义。根据 B 样条曲线/曲面的递归算法，在每两个节点的中点处

嵌入一个新节点，就会产生 25 个新的控制顶点。这些新的控制顶点就定义了四片新的子曲面。这样，对应于原控制网格中的每一个小四边形（如 $\boldsymbol{c}_{11}\boldsymbol{c}_{12}\boldsymbol{c}_{22}\boldsymbol{c}_{21}$），都有一个新顶点产生，称为面点；对于每一条边，也会有一个新顶点产生，称为边点；对于每一个顶点 $\boldsymbol{c}_{ij}$，也产生一个新顶点。这些新的面点、边点和顶点组成一个新的控制网格。该过程可重复进行下去，最终控制网格收敛到双三次 B 样条曲面。受此启发，Catmull 和 Clark 提出了面向任意拓扑控制网格的细分规则，具体包括如下两方面的几何细分和拓扑细分规则。

（1）几何规则。如图 3.9 所示，每个控制网格面在中心处加一个新的面点，如 Q_{11}；每条控制网格边加一个新的边点，如 Q_{12}；每个顶点用新的位置取代旧的位置，如 Q_{22}。



$$Q_{11}=(\boldsymbol{c}_{11}+\boldsymbol{c}_{12}+\boldsymbol{c}_{22}+\boldsymbol{c}_{21})/4$$
$$Q_{12}=((Q_{11}+Q_{13})/2+(\boldsymbol{c}_{12}+\boldsymbol{c}_{22})/2)/2$$
$$Q_{13}=(\boldsymbol{c}_{12}+\boldsymbol{c}_{13}+\boldsymbol{c}_{22}+\boldsymbol{c}_{23})/4$$
$$Q_{22}=\boldsymbol{Q}/4+\boldsymbol{R}/2+c_{22}/4$$

图 3.9　Catmull-Clark 细分的几何规则和拓扑规则（图片来自[8]）

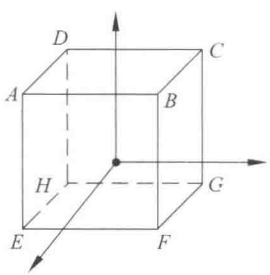
新顶点 Q_{22} 的计算公式中有 $\boldsymbol{Q}$ 和 $\boldsymbol{R}$ 两个新的变量，其中 $\boldsymbol{Q}=(Q_{11}+Q_{13}+Q_{33}+Q_{31})/4$，$\boldsymbol{R}=(1/4)((\boldsymbol{c}_{22}+\boldsymbol{c}_{12})/2+(\boldsymbol{c}_{22}+\boldsymbol{c}_{21})/2+(\boldsymbol{c}_{22}+\boldsymbol{c}_{32})/2+(\boldsymbol{c}_{22}+\boldsymbol{c}_{23})/2))$。

（2）拓扑规则。如图 3.9 中的虚线所示，新的边是由连接每个面点到邻接的边点以及连接每个顶点到邻接的边点所形成。这些新的顶点和边就组成了新的控制网格。

与 B 样条曲面采用矩形拓扑的控制网格不同，Catmull-Clark 细分曲面不受控制网格的拓扑限制，可作用到任意拓扑的控制网格上去。上述细分规则在作用一次以后，所有的面均变为四边形，而且从此以后度数不为 4 的顶点（称为奇异点）的个数会保持不变。除了奇异点以外，Catmull-Clark 曲面可以看作由一系列双三次 B 样条曲面覆盖而成，通过上述的细分规则就能够达到几乎处处连续的曲率。而在奇异点处，也能够保持切平面连续，但需要对细分规则做一些相应调整。

例题 3-5　Catmull-Clark 细分曲面。

问题： 右图所示的正立方体的 8 个顶点坐标分别是 $A(-1,1,1)$、$B(1,1,1)$、$C(1,1,-1)$、$D(-1,1,-1)$、$E(-1,-1,1)$、$F(1,-1,1)$、$G(1,-1,-1)$和 $H(-1,-1,1)$。写出采用 Catmull-Clark 细分生成新的控制网格顶点的伪代码。



解答： 假设 $k-1$ 次细分后顶点集合是 $V^{k-1}=\{v_1^{k-1},v_2^{k-1},\cdots\}$，其中 $V^0=\{A,B,C,D,E,F,G,H\}$，在 $k-1$ 次细分后所得的面和边的集合分别记为 $F^{k-1}=\{f_1^{k-1},f_2^{k-1},\cdots\}$ 和 $E^{k-1}=\{\boldsymbol{e}_{ij}^{k-1}\}$，那么第 k 次细分生成新的控制网格顶点的伪代码是

```
function CCSubdivision (V^{k-1}, F^{k-1}, E^{k-1})
    for each face f_{ijmn}^{k-1} = (v_i^{k-1}, v_j^{k-1}, v_m^{k-1}, v_n^{k-1}) in F^{k-1} do      /* 计算新的面点 */
        ṽ_{ijmn}^{k} ← (v_i^{k-1} + v_j^{k-1} + v_m^{k-1} + v_n^{k-1}) /4
    end for
    for each edge e_{ij}^{k-1} = (v_i^{k-1}, v_j^{k-1}) in E^{k-1} do                            /* 计算新的边点 */
        ṽ_{ij}^{k} ← ((v_i^{k-1} + v_j^{k-1}) /2 + (v_{ijmn}^{k} + v_{ijpq}^{k}) /2) /2          /* v_{ijmn}^{k} 和 v_{ijpq}^{k} 是面点 */
    end for
    for each vertex v_i^{k-1} in V^{k-1} do                                                      /* 计算新的顶点 */
        ṽ_i^{k} ← Q/4 + R/2 + v_i^{k-1} /4                                                       /* Q 和 R 是新变量 */
    end for
end function
```

□

3.3.2　Doo-Sabin 细分曲面

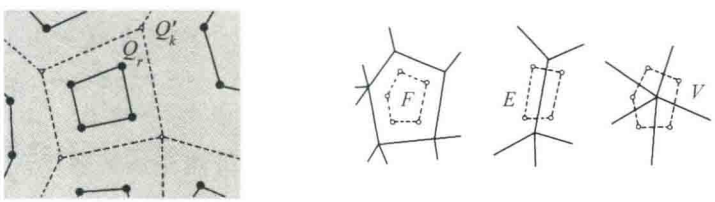
Doo-Sabin 细分曲面将双二次 B 样条曲面的生成方法推广到任意拓扑的控制网格，由 Daniel Doo 和 Malcolm Sabin 在 1978 年最先提出的。普通的双二次 B 样条曲面在递归计算过程中，每一个面上对应于每一个顶点，产生一个新顶点。连接这些新顶点，就会产生对应于原控制网格中的面点、边点和顶点的新面，并由此形成不断加密的控制网格。最终，这个加密过程得到的极限曲面就是双二次 B 样条曲面。受此启发，Doo-Sabin 细分采用如下两方面的几何细分和拓扑细分规则：

(1) 几何规则。每个控制网格面的 K 个顶点 $Q_1, Q_2, \cdots, Q_K$ 生成新的对应顶点 $Q_1', Q_2', \cdots, Q_K'$，其中

$$Q_k' = \sum_{j=1}^{K} \alpha_{ij} Q_j, \quad \alpha_{ij} = \begin{cases} \dfrac{K+5}{4K}, & i = j \\ \dfrac{3 + 2\cos(2(i-j)\pi/K)}{4K}, & i \neq j \end{cases} \tag{3.13}$$

这些新的顶点就定义了新的控制网格(如图 3.10(a)所示)。

(2) 拓扑规则。如图 3.10(b)所示，每个旧控制网格面的新顶点连接形成一个新的面 F；每条旧边两侧的 4 个新顶点连接形成新的面 E；每个旧顶点周围的新顶点连接形成新的面 V。



(a) 几何规划　　(b) 拓扑规则

图 3.10　Doo-Sabin 细分规则

经过一次 Doo-Sabin 细分后，每个顶点的度数均变为 4。再经过一次细分后，度数不为 4 的面的个数会保持不变。因此除了在有限个奇异点外，Doo-Sabin 细分曲面是由一

系列双二次 B 样条曲面覆盖而成。此外，Doo-Sabin 曲面在奇异点处也具有一阶光滑连续性。

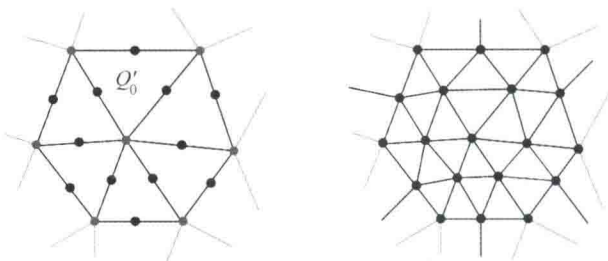
3.3.3 Loop 细分曲面

Loop 细分曲面将箱样条推广到三角形组成的控制网格，由 Charles Loop 在 1987 年最先提出。相比于 Catmull-Clark 细分曲面和 Doo-Sabin 细分曲面，Loop 细分曲面的细分规则较为简单。通过生成新的顶点，并依次将其连接形成新的控制网格（如图 3.11 所示）。具体来讲，Loop 细分采用如下几何和拓扑规则：

（1）几何规则。对于控制网格内部顶点 Q_0，假设其 N 个相邻顶点为 $Q_1, Q_2, \cdots, Q_N$，那么对应的新顶点 $Q'_0 = (1-N\beta)Q_0 + \beta\sum_{i=1}^{N} Q_i$，其中 $\beta = \left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{N}\right)^2\right)\Big/N$。对于控制网格边界上顶点 Q_0，假设与它相连的两个顶点是 Q_1 和 Q_2，那么对应的新顶点满足 $Q'_0 = \frac{3}{4}Q_0 + \frac{1}{8}(Q_1+Q_2)$。假设控制网格内部一条边的两个顶点是 Q_1 和 Q_2，相对的两个顶点是 Q_3 和 Q_4，那么新增加的顶点是 $Q'_0 = \frac{3}{8}(Q_1+Q_2) + \frac{1}{8}(Q_3+Q_4)$。如果 Q_1 和 Q_2 是控制网格的边界边上的两个顶点，则对应于该边界边所新增加的顶点是 $Q'_0 = \frac{1}{2}(Q_1+Q_2)$。

（2）拓扑规则。按照三角形控制网格的顶点和边的连接关系，将新顶点连接形成新的控制网格。

Loop 细分曲面除了一些特殊点外，几乎处处具有二阶光滑的连续性。

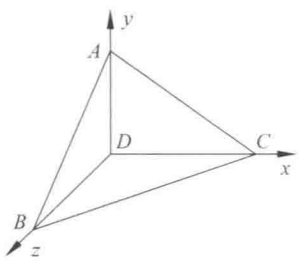


(a) 几何规则　　(b) 拓扑规则

图 3.11　Loop 细分规则

例题 3-6　Loop 细分曲面。

问题：右图所示正四面体的 4 个顶点的坐标分别为 $A(0,1,0)$、$B(0,0,1)$、$C(1,0,0)$ 和 $D(0,0,0)$。写出采用 *Loop* 细分生成新的控制网格顶点的伪代码。



解答：假设 $k-1$ 次细分后顶点集合是 $V^{k-1} = \{v_1^{k-1}, v_2^{k-1}, \cdots\}$，其中 $V^0 = \{A, B, C, D\}$。在 $k-1$ 次细分后所得的边的集合记为 $E^{k-1} = \{e_{ij}^{k-1}\}$，那么第 k 次细分生成新的控制网格顶点的伪代码是

```
function LoopSubdivision (V^{k-1}, F^{k-1}, E^{k-1})
    for each vertex v_i^{k-1} in V^{k-1} do
        if v_i^{k-1} in v_i^{k-1} ∈ V_int then                    /* 计算内部顶点 */
            ṽ_i^k ← (1 - Nβ) v_i^{k-1} + β Σ_{j=1}^{N} v_ij^{k-1}   /* {v_ij^{k-1}} 是相邻 N 个顶点 */
        else                                                      /* 计算边界顶点 */
            ṽ_i^k ← 3/4 v_i^{k-1} + 1/8 (v_i1^{k-1} + v_i2^{k-1})   /* v_i1^{k-1} 和 v_i2^{k-1} 是相邻顶点 */
        end if
    end for
    for each edge e_ij^{k-1} = (v_i^{k-1}, v_j^{k-1}) in E^{k-1} do
        if e_ij^{k-1} in E_int then                               /* 计算内部边界顶点 */
            ṽ_ijpq^k ← 3/8 (v_i^{k-1} + v_j^{k-1}) + 1/8 (v_p^{k-1} + v_q^{k-1})   /* v_p^{k-1} 和 v_q^{k-1} 是相对顶点 */
        else                                                      /* 计算边界边顶点 */
            ṽ_ij^k ← 1/2 (v_i^{k-1} + v_j^{k-1})
        end if
    end for
end function
```

□

3.3.4 Butterfly 细分曲面

Butterfly 细分曲面是由 Nira Dyn 等人于 1990 年提出的，主要针对三角形组成的控制网格。具体来讲，对于每个边，使用指定的规则创造一个新顶点，然后和旧的顶点把一个三角面片转化成四个新的三角面片（如图 3.12 所示）。Butterfly 细分采用如下几何和拓扑规则：

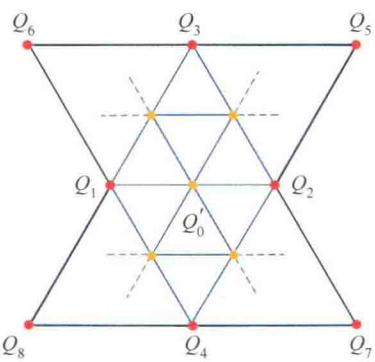


图 3.12 Butterfly 细分的几何规则和拓扑规则

（1）几何规则。对于三角形控制网格的每条边，利用可调控的权因子 ω 定义新的控制网格顶点 $Q'_0 = \frac{1}{2}(Q_1+Q_2)+\omega(Q_3+Q_4)-\frac{\omega}{2}(Q_5+Q_6+Q_7+Q_8)$。这里的权因子 ω 是可以根据细分曲面的外形要求而设定的。

（2）拓扑规则。依次将新的顶点和旧控制网格顶点连接，形成新的控制网格。

根据上述几何和拓扑规则得到的 Butterfly 细分曲面，除了一些特殊点外，几乎处处具有一阶光滑的连续性。

上述细分曲面在细分过程中使用的几何和拓扑规则不同，由此产生不同性质的细分曲面。图 3.13 展示了同一个控制网格，采用不同细分规则后所生成的细分曲面，可以看出其结果在形状和光滑性上都有着明显的差异。

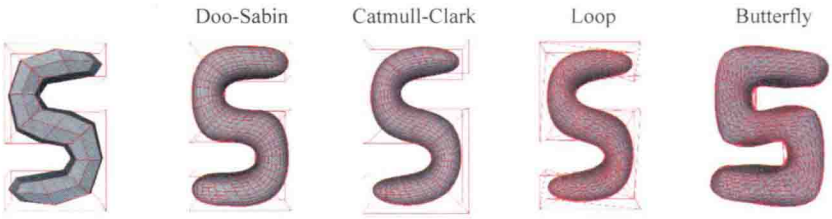


图 3.13 由相同控制网格生成的不同细分曲面

3.4 三维重建

自由曲线/曲面、细分曲面等几何建模方式是从数学模型出发来设计和构造几何形状，一定程度上要依赖设计人员关于几何形状的直觉和理解。与此相反，三维重建是直接或间接地获取真实世界中物体的形状和表观，使之能够在计算机中存储、处理和显示。因此，三维重建是一种从真实世界物体出发的几何建模方式。最具代表性的是 Stanford 大学的“米开朗基罗”项目。从 1998 年到 2000 年，该项目采用最先进的激光三维扫描仪将文艺复兴时代的著名雕塑作品全部进行几何建模，从而实现了文物的计算机数字化，起到了文物保护和文化传承的作用。

3.4.1 被动式与主动式建模

按照数据来源的不同，采用三维重建进行几何建模主要分为两种方式：被动式和主动式(如图 3.14 所示)。这两种都属于光学测量的范畴。其中，被动式包括基于图像的三维重建、基于视频的三维重建等；主动式包括基于激光测距的三维重建、基于 Kinect 的三维重建等。这两种方式各有优缺点，适用于不同的场合。

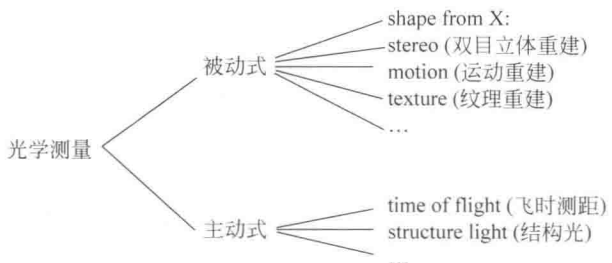


图 3.14 典型的三维重建方法

1. 被动式建模

被动式建模不需要与重建对象接触。在进行重建时，输入的数据是图像、视频等视觉信息，然后通过成像方式测量物体表面来推测三维结构。这类方法通常称为 Shape from X，X 可以指代轮廓、着色、纹理、阴影等能够通过设备获取的视觉信息。

被动式方法的优点是破坏性小、安全、成本较低。在数据采集时不需要和物体接触，所以不会对物体造成破坏，因此也避免了安全隐患。在建模时通常只需要根据拍摄的图

像、视频数据作为输入，因而成本较低。但缺点是对物体透明度敏感、不能处理镜面反射和内部折射，因此无法对玻璃材质的物体进行有效的几何建模。

2. 主动式建模

主动式建模是在采集数据时通过机械接触或主动观测等方式获取三维信息，例如传感器标记、结构光、激光、超声波等。按照扫描方式的区别可以分为三维扫描仪、飞时测距、三角测距、结构光等。

接触式三维扫描仪，如坐标测量机等，具有测量精确度高的优点，可达到微米级别。但是这种方式价格昂贵，且需要专业的操作者。飞时测距，是通过发出激光脉冲，并计算这束光返回所需要的时间来测算距离。这种方式的优点是扫描速度快、便携、方便，而且测量范围大；但缺点是精度有限，只能达到毫米级别。三角测距，是通过发射一道激光到待测物上，并利用摄影机记录待测物上的激光光点，然后利用激光光点、摄影机、激光源构成的三角形来测算距离。这种方式的优点是精度较高、适合测量大尺寸物体；但缺点是扫描速度慢，需要花费较长时间来完成三维重建。结构光扫描，如 Kinect 等，则使用红外线发射器发射红外光线，然后利用红外线传感器接收反射回来的红外光线来获取深度图像。这种方式的优点是设备价格便宜、易于安装，但缺点是精度较差，而且测量范围有限（40cm～3.5m）。

3.4.2 基于图像的三维重建

基于图像的三维重建（Image-based reconstruction，IBR），是指从拍摄的图像对三维物体的外形进行重建。按照输入图像数量的不同，可分为多视角重建和单幅图像重建。多视角重建需要输入多幅图像，然后恢复物体的外形。单幅图像重建仅需要一幅图像作为输入进行重建。一般情况下，图像像素记录了来自不同方向的入射光信息。因此，增加输入图像数量能够提升几何模型重建的精度。

如图 3.15 所示，多视角重建的算法流程一般包括四个步骤：摄像机标定、三角测量、从运动恢复结构（稀疏形状估计）以及立体匹配（稠密形状估计）。

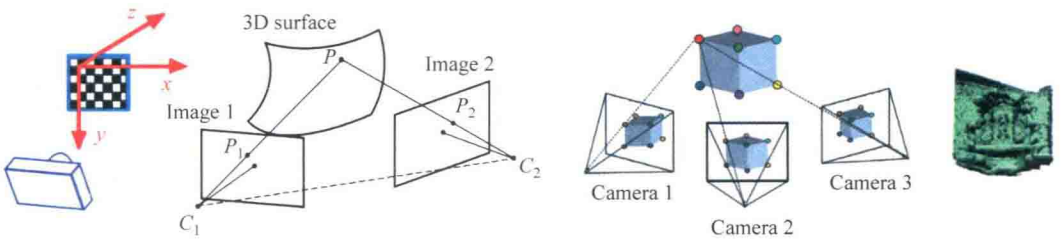


图 3.15 多视角三维重建算法流程

1. 摄像机标定（Calibration）

该步骤从一些已知坐标的三维空间点集，反求相机的内参（如焦距等）和外参（相机的姿态）变量，进而得到与物体三维信息有关的相机运动。摄像机的成像模型通常简化为小孔成像模型，那么从三维空间到二维成像平面的成像可以看作投影变换。因此，摄像机的

成像模型表示为：

$$\boldsymbol{x}_{3\times1}=\boldsymbol{P}_{3\times4}\cdot\boldsymbol{X}_{4\times1}\Rightarrow\begin{bmatrix}x\\y\\1\end{bmatrix}=\begin{bmatrix}p_{11}&p_{12}&p_{13}&p_{14}\\p_{21}&p_{22}&p_{23}&p_{24}\\p_{31}&p_{32}&p_{33}&p_{34}\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}\tag{3.14}$$

其中，$\boldsymbol{x}$ 是二维成像平面上的像素坐标，$\boldsymbol{X}$ 是三维空间中的点坐标，$\boldsymbol{P}_{3\times4}$ 是描述小孔成像过程的投影矩阵。为了方便统一的表示，这里采用齐次坐标形式。摄像机标定的主要任务就变成计算投影矩阵 $\boldsymbol{P}_{3\times4}$，进一步可以分解为内参矩阵 $\boldsymbol{K}_{3\times3}$ 和外参矩阵 $\boldsymbol{M}_{3\times4}$，即 $\boldsymbol{P}_{3\times4}=\boldsymbol{K}_{3\times3}\cdot\boldsymbol{M}_{3\times4}$。

内参矩阵描述了在摄像机坐标系下，三维空间到二维图像的投影变换。假设摄像机位于原点，朝向与坐标轴重合，而图像坐标系的原点在图像中心，那么内参矩阵 $\boldsymbol{K}_{3\times3}$ 可以表示为一个对角矩阵和上三角矩阵的乘积：

$$\boldsymbol{K}_{3\times3}=\begin{bmatrix}m_x&&\\&m_y&\\&&1\end{bmatrix}\begin{bmatrix}f&&p_x\\&f&p_y\\&&1\end{bmatrix}\tag{3.15}$$

其中，m_x 和 m_y 是由于透镜弯曲产生的畸变系数，f 是焦距，(p_x,p_y) 是图像中心的像素坐标。外参矩阵则描述了摄像机姿态的变化对不同视角成像的影响。通常把摄像机看作刚体，其姿态变化就可以通过旋转和平移变换来实现，也就得到 $\boldsymbol{M}_{3\times4}=[\boldsymbol{R}|\boldsymbol{t}]$。其中 $\boldsymbol{R}_{3\times3}$ 是三维空间中的旋转矩阵，$\boldsymbol{t}$ 是平移向量。通过该矩阵，就可以描述摄像机在拍摄两幅图像时的空间位置关系。

而在实际中，不同视角图像拍摄时摄像机姿态是不同的，这就需要直接根据多幅图像之间的对应关系计算相应的内参和外参矩阵。这个过程通常包含 4 个步骤：角点检测、投影变换计算、参数估计、参数优化。

角点检测是为了获取图像中具有视觉显著性的特征点，然后通过特征点匹配建立不同图像之间的对应关系。为了提高检测和匹配的准确性，通常采用固定模式的物体做参考图像，例如棋盘格。首先进行边缘检测，将各个矩形框边缘拟合成直线；然后，计算直线交点作为角点位置。这样通过角点所在直线的行列排布，就可以直接建立不同图像之间角点的对应关系。进一步，将棋盘格摆放在摄像机前的指定位置，就能获得这些角点在三维空间中的坐标。

投影变换计算则是通过角点的二维图像坐标与三维空间坐标求解变换矩阵 $\boldsymbol{P}$。常用的计算方法有直接线性变换法(Direct linear transform，DLT)、最小二乘优化等，其本质是优化三维空间点在投影变换后与图像角点的几何误差。

参数估计是通过对投影变换的分解，估计内参矩阵和外参矩阵的初始值。根据内参矩阵和外参矩阵的形式，将 3×4 的投影变换矩阵 $\boldsymbol{P}$ 表示为 3×3 的矩阵 $\boldsymbol{B}$ 和 3×1 的向量 $\boldsymbol{b}$，也就是写成 $\boldsymbol{P}=[\boldsymbol{B}|\boldsymbol{b}]$，那么，内参矩阵可以表示为 $\boldsymbol{K}=\boldsymbol{B}\cdot\boldsymbol{B}^{\mathrm{T}}$。进一步，对 $\boldsymbol{B}$ 进行 QR 分解可以得到矩阵 $\boldsymbol{R}$ 和 $\boldsymbol{Q}$，其中 $\boldsymbol{R}$ 就对应于外参矩阵的旋转变换，平移变换对应的向量则可以通过 $\boldsymbol{t}=\boldsymbol{Q}^{-1}\cdot\boldsymbol{b}$ 来计算。

上述参数估计是对单张图像的角点投影误差进行优化求解，而不同视角下的内参矩

阵需要保持一致。另一方面，通过矩阵分解计算得到的内外参数，本质上是通过优化代数距离进行的计算，并没有实际的物理意义。这也就是说，获得的解是有几何偏差的。因此，需要对多视角投影误差做进一步优化，以便进一步提高摄像机标定的准确性。

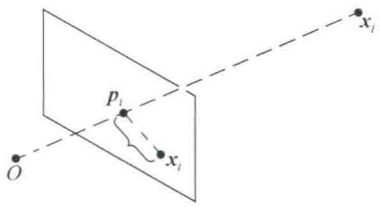


图 3.16 重投影误差

给定多视角下 n 组对应角点的二维图像坐标和三维空间点坐标，计算角点检测位置与其通过投影变换模型预测的成像点之间的重投影误差，将其做最小化处理(如图 3.16 所示)，建立关于矩阵元素的优化模型。该过程可以通过非线性最小二乘优化进行计算，也就是求解能量函数 $E(\boldsymbol{K},\boldsymbol{R},\boldsymbol{t}) = \min_{P}\sum_{i} \| x_i - \boldsymbol{K}[\boldsymbol{R} \quad \boldsymbol{t}]\boldsymbol{X}_i \|^2$ 的局部最优解。数学上，这个优化问题可以使用 LM 算法(Levenberg-Marquardt algorithm)进行有效求解。

2. 三角测量(Triangulation)

通过指定棋盘格图像对摄像机进行标定后，就可以根据任意拍摄的图像对物体进行三维重建。首先需要通过标定的内参和外参矩阵，并结合两幅或多幅图像间特征点的对应关系，计算相应的三维空间坐标。三角测量是一种最简洁的方法。它利用已知空间点 $\boldsymbol{X}$ 在两幅图像上的投射点的坐标和两条投射线与基准线的夹角，估计这些点在三维空间中的位置。基准线是指两个相机坐标系原点的连线，那么形成的三角形的第三个点就是点 $\boldsymbol{X}$ 的三维坐标。但是由于成像噪声及计算误差的原因，两条直线在三维空间中可能不会产生严格意义的相交。这时，可通过求解距离两条射线最近的点作为近似交点。

3. 从运动恢复结构(Structure-from-motion，SFM)

将摄像机标定和三角测量的结果作为初始值，通过分析物体在不同视角成像的运动，就可以得到更加准确的三维模型的结构信息。这一过程通常是利用同一个三维空间点投影到不同视角图像上二维特征点的投影关系来实现，同时也得到摄像机参数和三维空间坐标。

光速平差法(Bundled adjustment)是常用的方法之一。它是基于三维模型结构和视角参数(如相机位置、朝向、固有标定和径向畸变)的优化问题，用于获得最佳的三维坐标和运动(如相机矩阵)参数估计。这种方法本质上是最小化投影变换所决定的重投影误差。具体来讲，对应于如图 3.17 所示的一个空间点的重投影误差，可以采用如下公式进行优化求解：

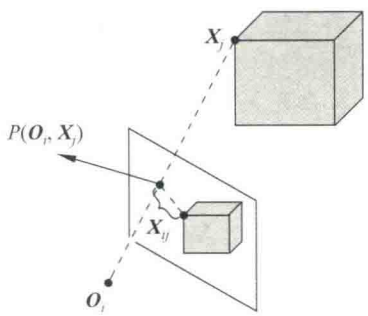


图 3.17 光束平差法的重投影误差

$$\min \sum_{1}^{n}\sum_{1}^{m}\omega_{ij} \| \boldsymbol{X}_{ij} - P(\boldsymbol{O}_i,\boldsymbol{X}_j) \|^2 \qquad (3.16)$$

其中 $\boldsymbol{X}_{ij}$ 表示三维空间中的点 $\boldsymbol{X}_j$ 在第 i 个相机投影后的像素位置，ω_{ij} 是和相机位置分布有关的权因子，$P(\cdot,\cdot)$表示对应的相机投影操作。公式(3.16)可以使用最小二乘算法来进行最小误差的计算，最终得到最优的投影矩阵和三维模型。

4. 立体匹配(Stereo match)

通过从运动恢复结构,可以得到更精确的摄像机参数和特征点的三维空间位置。然而,由特征点重建的形状通常是稀疏的点云。这主要是由于能够作为特征点的像素只是占图像的很少一部分。因此,需要对所有的像素点同时应用特征匹配标准来获取它们的三维深度值,从而得到更加稠密的三维点云模型。这个过程就是立体匹配。

极线约束是在立体匹配时常用的方法。图 3.18 展示了极线约束的求解方法。具体来讲,假设空间中的点 X,在两个相机成像平面上的投影点分别为 $\boldsymbol{x}_{\mathrm{L}}$ 和 $\boldsymbol{x}_{\mathrm{R}}$。这里,$\boldsymbol{O}_{\mathrm{L}}$ 和 $\boldsymbol{O}_{\mathrm{R}}$ 分别为两个相机的中心,也就是相机坐标系的原点。在极线几何中,$\boldsymbol{O}_{\mathrm{L}}$ 和 $\boldsymbol{O}_{\mathrm{R}}$ 的连线称为基线。基线和两个相机成像平面的交点 $\boldsymbol{e}_{\mathrm{L}}$ 和 $\boldsymbol{e}_{\mathrm{R}}$ 分别为极点。它们分别是两个相机中心 $\boldsymbol{O}_{\mathrm{L}}$ 和 $\boldsymbol{O}_{\mathrm{R}}$ 在对应的成像平面上的投影点。$\boldsymbol{X}$、$\boldsymbol{O}_{\mathrm{L}}$ 和 $\boldsymbol{O}_{\mathrm{R}}$ 组成的三角形所在的平面称为极平面 π。在极平面上另外取一个点 $\boldsymbol{X}_1$,那么它在两个相机平面上的投影点分别是 $\boldsymbol{x}_{\mathrm{L}}$ 和 $\boldsymbol{x}_{\mathrm{R1}}$。这里,$\boldsymbol{x}_{\mathrm{R1}}$ 和 $\boldsymbol{x}_{\mathrm{R}}$ 都在极线 l_{R} 上。这种共线性质就是极线约束。因此,当给定一点后,它的匹配点一定出现在它所对应的极线上。根据极线约束,可以将搜索空间压缩到一维的极线上。在求得极线后,对图像上沿极线方向上的像素点按照灰度相似性进行匹配,能够很方便地找出该点在对应图像上的匹配点,从而恢复出该像素所对应的三维空间位置。

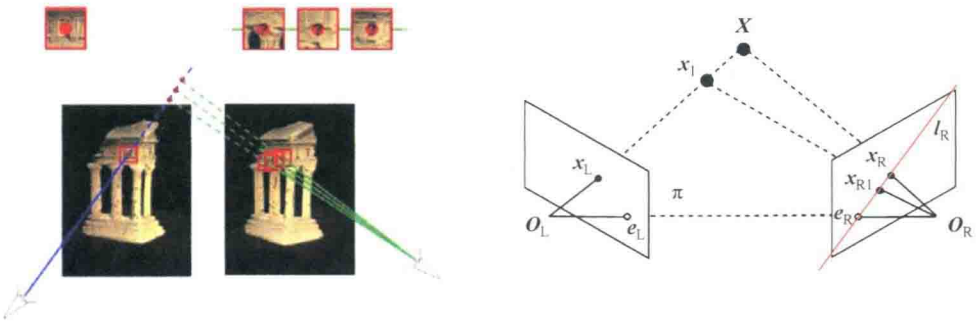


图 3.18 立体匹配中的极线约束(图片来自[10])

从上述过程可以看出,多视角重建本身其实是一个病态数学问题。然而,通过摄像机标定等方式可以增加额外约束并获得初始解。这样就能够使得问题变得可以求解。从单幅图像进行三维重建则是一个高度病态的问题,因为这种情况下可用来求解的约束条件变得更加稀少。为了实现单幅图像的三维重建,需要引入更多的先验知识来增加约束条件。人工交互的方式通常认为是增加先验知识的有效手段。此外,也可以结合物体对象本身几何和物理属性,如平面、线结构、对称性等,在人工交互的基础上智能地完成三维重建。

3.4.3 基于视频的三维重建

与基于图像的三维重建不同,基于视频的三维重建是将视频帧序列作为输入,然后进行几何建模。但是,二者本质上都是采用图像或视频帧提供的视觉信息来估计三维形状。基于视频重建的优点是数据容易拍摄,可以提供更加完整的信息,因而具有较大的灵活

性。但缺点是视频拍摄中不可避免地产生冗余数据。如果不能很好地处理帧序列的时空关系，重建的结果容易产生结构混淆等问题。

基于视频的三维重建主要分两种思路：基于关键帧的三维重建和基于深度图恢复的三维重建。前者从输入的视频序列抽取若干关键帧，然后利用基于图像的建模方法进行三维重建；后者则是通过计算每个像素点的深度值，恢复物体的整体三维形状。

1. 基于关键帧的三维重建

这种方式抽取帧序列中的子集作为关键帧图像，然后采用多视角三维重建。因此，这里的核心问题是如何选择关键帧。通常利用几何鲁棒性标准(GRIC)：根据关键帧之间特征点的对应关系计算基础矩阵 $\boldsymbol{F}$，同时利用矩阵分解得到单应变换 H。进一步，比较哪个变换产生的误差小，也就是分数低。如果单应变换得到的分数低，则相对于第一张图像来说，第二张图像不能作为参考帧。依此类推，从帧序列里面抽取出合适的关键帧。接下来，按照基于图像三维重建的方法实现对视频中物体三维模型的重建。

2. 基于深度图恢复的三维重建

这种方式需要解决的一个重要问题是如何保持相邻帧深度图恢复的时空一致性，进而重建出有效的三维形状。通常做法是首先利用 SFM 得到摄像机参数，也就是对应于每一帧的内参和外参矩阵。然后根据外参矩阵估计帧间连续变化的初始深度分布，通常表示为相邻帧的视差。这个视差反映了相邻帧的深度差异。最后，通过帧间匹配特征点的极线约束，对初始深度进行迭代优化，从而得到更准确的深度图。在此基础上，通过多面体逼近三维点云的形状，得到由多边形面片组合而成的表面连续的三维模型。这种多边形组成的多面体又称为网格(Mesh)，而从点云到网格的转化过程则称为网格化。网格的相关概念将在第 4 章中具体介绍。

3.4.4 基于激光测距的三维重建

这种方式是利用激光测距工作的原理，通过记录被测物体表面大量的、稠密的三维坐标、反射率和纹理等信息，直接获取物体的三维模型。

基于激光测距的三维重建首先使用三维扫描仪采集物体表面的点云数据，并通过视点规划来选择合适的位姿以消除由于遮挡所导致的点云空洞。然后，把从多个视点扫描得到的点云数据经过配准转换到同一个坐标系下，合并成为一个完整的物体表面点云模型。接着，利用点云重建算法得到连续的网格形式的三维模型。最后，结合纹理映射等方式可以进一步对模型添加颜色等信息。对于数据的扫描获取，已经可以通过成熟的硬件和软件实现。现阶段三维重建的主要问题是从离散点云到连续网格的转换，也就是点云网格化。

数学上，网格化是采用 C^0 连续的多面体插值或者拟合离散的点云，从而精确地或近似地表现物体表面的三维形状。经典的网格化方法有 Marching cube 方法、Delaunay 三角化方法、移动最小二乘方法、泊松方法等。

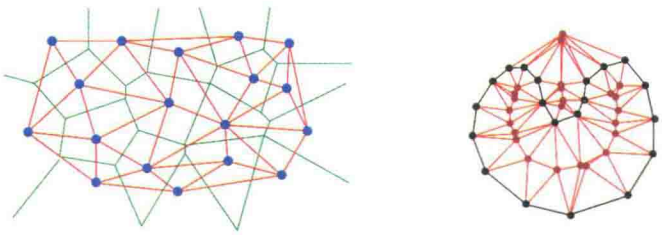
1. Marching cube 方法

该方法通过构造三维空间中的函数等值面进行网格化重建，由美国 GE 公司的 William Lorensen 和 Harvey Cline 于 1987 年最早提出。该方法早期处理的对象一般是

断层扫描(CT)、核磁共振成像(MRI)等产生的三维图像，但对于后期出现的三维激光扫描数据也同样适用。这种方法在给定采样得到的一个三维离散数据集后，将其分割为多个小立方体，那么相邻的 8 个采样点则会构成一个立方体。然后，按扫描线顺序在每个立方体中构建等值面，以此作为采样点的三维重建表面，将所有等值面相连得到完整的三维网格。

2. Delaunay 三角化

该方法利用计算几何中的 Delaunay/Voronoi 图进行点云的网格化。Delaunay 图是对离散点集进行三角剖分的一种方法，所形成的三角形具有最大化最小角等优良性质。Voronoi 图则是 Delaunay 图的对偶图：每个 Delaunay 三角形的几何中心作为 Voronoi 图的顶点；相邻 Delaunay 三角形几何中心的连线作为 Voronoi 图的边(如图 3.19(a)所示)。在对三维空间点云数据网格化时，首先计算采样点的三维 Voronoi 图，其顶点作为相应采样点的极点。然后，连同极点和采样点再做一次 Delaunay 三角剖分。图 3.19(b)是在二维平面上对离散点集做三角剖分的示意图。根据这次三角剖分的结果，从中抽取出三个顶点都是采样点的三角形。这样就形成了离散的三角形面片集合，得到网格化结果。



(a) Delaunay三角化（三角形）和Voronoi图（多边形）　　(b) 基于Delaunay三角化的网格

图 3.19　Delaunay 三角化

3. 移动最小二乘拟合

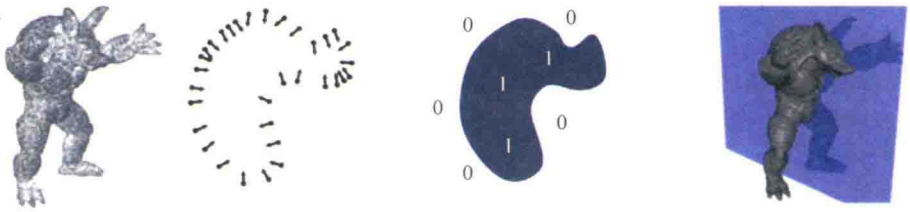
该方法利用移动最小二乘函数来对三维点云数据进行函数逼近，其本质是一种基于局部多项式逼近的方法。假设三维空间中的 N 个点，如果采用加权平方距离多项式作为逼近函数，那么就可以定义如公式(3.17)所示的移动最小二乘函数：

$$\sum_{i=1}^{N}[f_x(x_i)-\boldsymbol{P}_i]^2 w(||x_i-x||) \tag{3.17}$$

其中，w 是和点之间距离相关的权重函数，$\boldsymbol{P}_i$ 表示三维空间的点。通过该函数可以度量逼近函数和插值点的几何误差，从而借助几何误差最小化来找到对应每个点的最优函数 f_x。移动最小二乘拟合的重建表面具有比较好的光滑性，而且对于局部几何细节的重建也相对准确。

4. 泊松方法

该方法利用泊松方程表示物体的三维形状。具体来讲，给定一个有向点集(具有法向信息的三维点云)，使用隐函数框架来计算一个三维指示函数。该函数在模型内部的点取值为 1，外部的点取值为 0，那么该函数对应于泊松方程的解。通过数值求解该方程，并提取 0 和 1 之间的等值面来获取重建的三维形状(如图 3.20 所示)。泊松重建的网格一定是封闭的，在其表面可以保证不出现空洞。



(a) 有向点集　　(b) 指示函数　　(c) 三维重建

图 3.20　基于泊松方法的三维重建(图片来自[13])

3.4.5　基于 Kinect 的三维重建

基于 Kinect 的三维重建是一种典型的基于结构光的主动式方法。Kinect 是 Microsoft 公司在 2010 年左右推出的体感设备，最早用于 Xbox 游戏操作。它提供了一种基于人体姿态的非接触式交互方式。作为该设备的核心功能，Kinect 搭载有一个彩色摄像头、一个红外发射器以及一个红外摄像头(如图 3.21 所示)。该设备能够实现场景深度的实时捕捉。

图 3.21　Kinect 设备及拍摄的彩色和深度图像(图片来自[14])

红外发射器发射激光散斑，那么空间中不同位置的散斑成像图案都是不同的。只要在三维空间中发射这样的结构光，整个空间就都被做了标记。如果把一个物体放入空间，通过拍摄物体表面上的散斑图案，那么就可以推测其空间位置，也就是获取了场景深度。由于深度测量的精度与物体离 Kinect 的实际距离成反比，当距离超过 3 米时，获取的深度精度会大大降低。因此，Kinect 仅适用于室内环境下体积较小物体的三维重建。

KinectFusion 是目前广泛使用的基于 Kinect 的三维重建方法。该方法采用一台围绕物体移动的 Kinect，实时重建物体的三维模型。不同于深度图的简单拼接，KinectFusion 方法的特性在于：如果对物体进行持续的扫描，三维重建精度可以由粗到细、自适应地逐渐提高。该方法主要包括如下四个步骤：深度图转换、姿态配准、立体空间融合、模型绘制。

1. 深度图转换

该步骤将 Kinect 中获取的原始深度帧数据转换为和 Kinect 摄像头朝向一致的点云数据，也就是转换到相机坐标系中。由于 RGB 摄像机和红外摄像头在同一个载体上，利用 RGB 相机的内参矩阵进行变换即可实现转换。同时也将 RGB 彩色图的像素和深度图

的深度信息建立了关联，从而获得法向、纹理等信息。

2. 姿态配准

该步骤将不同帧对应的深度图进行融合。数学上，这是计算不同深度图之间的刚体变换，使得变换后的深度和其他帧上对应点的深度相吻合。目前大多采用经典的迭代最邻近点优化(Iterative closest point，ICP)方法。

3. 立体空间融合

根据深度图中的像素深度，构造体积分模型。数学上，这可以表示为像素平面作为定义域、深度值作为函数值的二重积分运算，从而生成封闭的体模型。

4. 模型绘制

基于光线跟踪的思想，通过投射光线与体模型的相交运算，得到模型表面对应交点位置、法向等参数。然后，利用光线跟踪方法对模型进行真实感绘制。同时，结合捕获的颜色等信息，使得绘制的模型外观更加真实。

3.5 其他建模方法

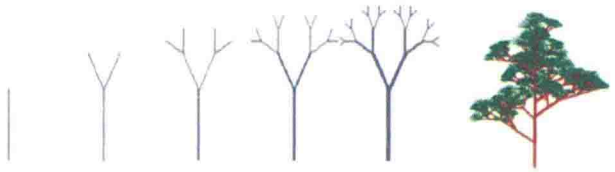
自由曲线/曲面、细分曲面、三维重建等提供了有效的几何建模手段，具有各自的优点和不足。然而真实世界中的物理形状多样，对于一些特殊形状，是可以采用更有针对性的建模方法的。接下来，介绍以分形、粒子系统、语言学为代表的其他几何建模方法。

3.5.1 分形

分形(Fractal)，是指一种具有以非整数维形式充填空间的形态。从整体上看，分形是处处不规则的，例如海岸线、珊瑚等。虽然这类形状从远距离观察是极不规则的，但是不同尺度上的规则性又是相同的。例如从近距离观察海岸线和珊瑚，其局部形状又和整体形态极为相似。因此，这类形状从整体到局部，都是自相似的。分形一词，最早由法国数学家 Benoit Mandelbrot 在 1975 年提出。而分形作为一类几何问题，最早起源于他在 1967 年发表在《科学》上的论文，用于研究英国的海岸线形状。

L 系统是一种典型的分形建模方法，主要用于花草树木类形状的几何建模。L 系统最早是由匈牙利生物学家 Aristid Lindenmayer 在 1968 年为模拟生物形态而设计的。1984 年，Lucas 电影公司的 Aluy Smith 将 L 系统应用于计算机图形学，用于复杂形状的几何建模。L 系统以自动机理论为基础，用符号空间的一个符号序列来表示状态，通过符号序列的变化来描述形态生长的过程，从而生成复杂的几何形状。图 3.22(a)是一个基于 L 系统的树木建模实例。

另一种分形建模方法是中点位移法，主要用于地形地貌几何建模。这种方法的数学基础是谢尔宾斯基三角形(Sierpinski triangle)，通过各边中点的随机位移生成地形起伏。该方法通常基于一个实心的三角形(多数使用等边三角形)；然后沿三边中点的连线分成四个小三角形；接着去掉中间的那一个小三角形；最后对其余三个小三角形重复这个过程。中点位移能够随机生成地形高度，体现地貌起伏变化。图 3.22(b)是基于中点位移法的地形建模实例。



(a) 基于L系统的树木建模

(b) 基于中点位移法的地形建模

图 3.22　基于分形的几何建模

3.5.2　基于粒子系统的建模

与固体形态物体的几何形状不同，火焰、烟雾、雨、雪、云等的形状呈现一定的流体性，但其物质构成却是颗粒状的微小物体。这类形状通常采用粒子系统进行几何建模。粒子系统是利用物理学中微观粒子的运动，模拟大量的、按一定规则运动（变化）的微小物质组成的大物质。这种方式既可以呈现流体特性，又保持了固体的微小物体组成。

如图 3.23 所示，粒子系统主要由两部分组成：粒子图元和发射器。粒子图元，是指具有一定生命周期和自身属性的微小个体模型，记录形状、大小、颜色、透明度、运动速度和方向、生命周期等。发射器，则是作为粒子的源，并控制它们的产生和状态。利用粒子系统进行几何建模时的步骤主要有五步：产生新的粒子加入系统；赋予每个粒子一定属性；删除超过生命值上限的粒子；根据粒子属性的动态变化进行更新；绘制并显示具有生命的粒子。

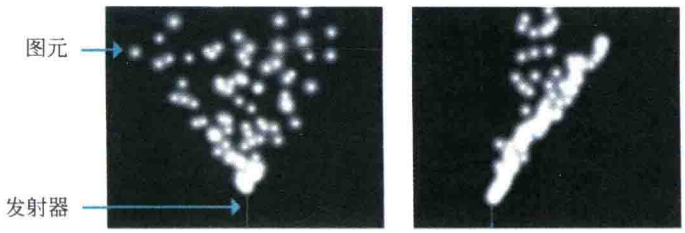
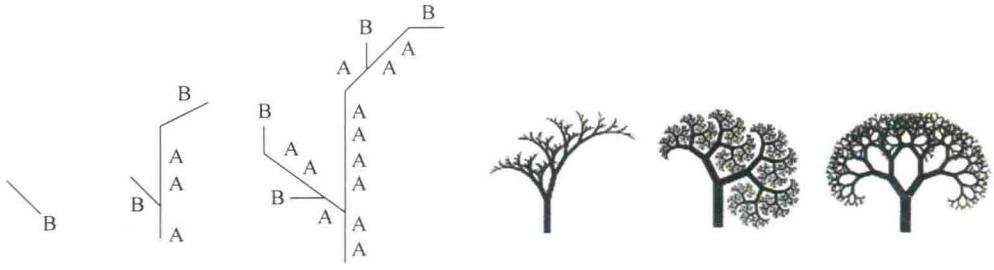


图 3.23　粒子系统建模

3.5.3　基于语言学的建模

这种建模方式主要是模拟人类语言系统的组成，通过一组符号及其相应语法对物体形状进行程序化描述。以图 3.24 所示的植物建模为例，首先定义语言规则，包括基本的

符号集合(如 A 和 B 两个元素,以及一些括号表示运算顺序)、两个符号之间的运算规则(即语法,用于指明元素之间的相互作用)。然后,根据语言规则及植物形状的特性,定义相应的几何操作(例如,图 3.24(a)中的 A 是主干形状,B 是分枝形状)。最后,根据相应的主干和分支顺序,便可以生成相应的几何模型(如图 3.24(b)所示)。



(a) 语言　　(b) 植物

图 3.24　由语言规则定义的植物建模

3.6　几何数据结构

计算机内部需要采用合适的数据结构对几何建模的过程和结果进行表示和存储。这里的数据结构是指在计算机内部表示、存储几何模型的数据组织形式。常见的数据结构有层次模型、构造实体几何模型、八叉树模型、边界模型等,需要根据实际的应用场合选择合适的数据结构。其中,层次模型、构造实体几何模型、八叉树模型属于实体造型中的常用数据结构,侧重于建模的规则,以及模型的物理真实程度,而不仅仅是视觉和仿真上的精确性。边界模型则主要是面向计算机动画、计算机仿真等应用中的三维形状,追求视觉上的真实感。接下来,介绍构造实体几何模型和边界模型两种典型的数据结构。

3.6.1　构造实体几何模型

构造实体几何模型(Constructive solid geometry,CSG),是通过元几何体素及其布尔操作来描述三维形状,从而把复杂实体构造转化为一些有序的简单实体的布尔运算。这里的元几何体素包括长方体、球体、圆柱体、圆锥体、圆环体等,布尔操作则包括并、交、差等正则集合运算,分别用符号∪、∩、－表示。图 3.25 展示了两个体素在不同布尔操作下的运算结果。

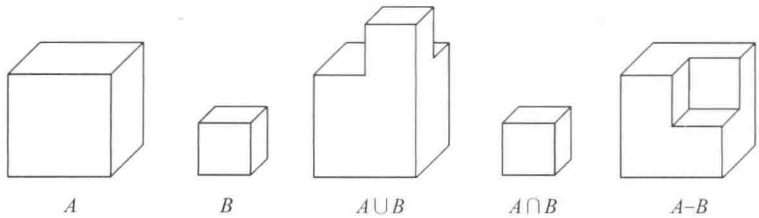


图 3.25　元几何体素的布尔运算结果

构造实体几何模型的优点是具有非常高的数学精确性，能够有效表示固体或者密闭物体的模型，因此广泛应用于 CAD/CAM 等系统。但其缺点也很明显，表示形式复杂，而且表示的物体大多是具有规则外形的机械零件等。

3.6.2 边界模型

这是一种采用几何元素列表描述多面体网格的数据结构，又称为边界表示法(B-rep)。与构造实体几何模型不同，边界表示法显式地记录了构成模型的外形信息，一般采用顶点、边、面等作为基本几何元素。其中，简单边界模型和半边模型是两种典型的数据结构。

1. 简单边界模型

简单边界模型只记录最基本的顶点、面的信息。其中，顶点信息可以包括空间坐标、纹理坐标、法向等，而面信息主要记录构成一个多边形面片的顶点序列。以图 3.26 中的三角网格为例，它是由三角形面片组成的多面体。对于每个顶点 $\boldsymbol{v}$，记录其三维空间坐标 (v_x, v_y, v_z)。如果三维模型具有纹理和法向信息，那么也可以记录相应的二维纹理坐标 (t_x, t_y)和三维法向(n_x, n_y, n_z)。对于每个三角形面片，记录其三个顶点的索引$\{v_1, v_2, v_3\}$。如果三维模型具有纹理和法向信息，那么也可以追加三个顶点对应的纹理坐标和法向的索引，即$\{v_1, v_2, v_3, t_1, t_2, t_3, n_1, n_2, n_3\}$。

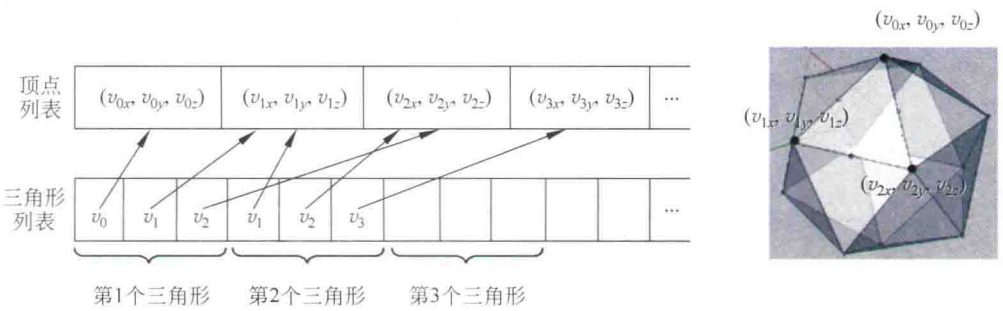
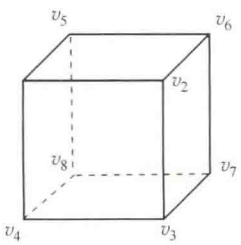


图 3.26 简单边界模型

简单边界模型最常见的数据结构是由一个共享的顶点列表和一个存储指向它的顶点的指针列表构成。简单边界模型的优点是简单、方便构造；但其缺点也明显，复杂数据操作困难，尤其体现在不容易获得属于不同面片的顶点、边等几何元素之间的相邻关系，以及快速地执行插入/删除操作。

例题 3-7 三维物体的简单边界模型数据结构。

问题： 如右图所示单位立方体，假设 8 个顶点坐标分别是 $v_1(0,1,1)$，$v_2(1,1,1)$，$v_3(1,0,1)$，$v_4(0,0,1)$，$v_5(0,1,0)$，$v_6(1,1,0)$，$v_7(1,0,0)$，$v_8(0,0,0)$。那么，对于顶点和面分别采用一维数组的数据结构表示该立方体的简单边界模型。



解答： 单位正方体 8 个顶点对应一维数组 float $V[24]$，依次记录顶点的 x、y、z 坐标分量，定义为：

float $V[24]=\{0,1,1,1,1,1,1,0,1,0,0,1,0,1,0,1,1,0,1,0,0,0,0,0\}$;

此外，该正方体 6 个面都是四边形，分别是 $f_1(v_1,v_2,v_3,v_4)$，$f_2(v_2,v_6,v_7,v_3)$，$f_3(v_5,v_6,v_7,v_8)$，$f_4(v_1,v_5,v_8,v_4)$，$f_5(v_1,v_2,v_6,v_5)$，$f_6(v_4,v_3,v_7,v_8)$。那么，可以采用一维数组 int F[24]，依次记录每个面的顶点在数组 V 中的 x 坐标索引，定义为：

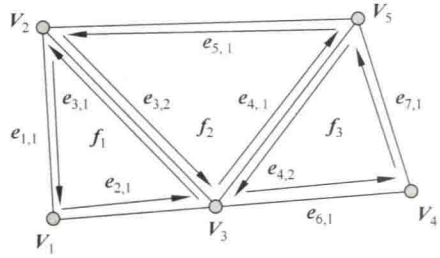
int $F[24]=\{0,3,6,9,3,15,18,6,12,15,18,21,0,12,21,9,0,3,15,12,9,6,18,21\}$;

这里，第 n 个顶点对应的 x 坐标在数组 V 中的索引为 $(n-1)\times 3$。通过 V 和 F 两个数组，就可以直接获得立方体的每个面所包含的顶点及其坐标信息。

□

2. 半边模型

半边模型的主要特点是将三维网格的一条边表示成两条具有相反指向的半边。因此，对应于同一条边的两个半边具有相反的方向(如图 3.27 所示)。一般情况下，半边记录相应的顶点、下一个半边、前一个半边、相邻面片等信息。因此，从半边模型可以很容易得到各种几何元素之间的相邻关系。以图 3.27 为例，基于半边模型的数据结构可以使用顶点或半边作为主要索引。其中，如图 3.27(b)所示，以顶点为主要索引的半边数据结构记录顶点的三维空间坐标、指向的半边等信息。以半边作为主要索引的半边数据结构记录开始顶点、相反半边、下一个及前一个半边等(如图 3.27(c)所示)。



(a) 半边结构

顶点	坐标	开始点的半边
V_1	(v_{1x}, v_{1y}, v_{1z})	$e_{2,1}$
V_2	(v_{2x}, v_{2y}, v_{2z})	$e_{1,1}$
V_3	(v_{3x}, v_{3y}, v_{3z})	$e_{4,1}$
V_4	(v_{4x}, v_{4y}, v_{4z})	$e_{7,1}$
V_5	(v_{5x}, v_{5y}, v_{5z})	$e_{5,1}$

(b) 顶点列表

半边	开始点	相反半边	三角形	下一个半边	前一个半边
$e_{3,1}$	v_3	$e_{3,2}$	f_1	$e_{1,1}$	$e_{2,1}$
$e_{3,2}$	v_2	$e_{3,1}$	f_2	$e_{4,1}$	$e_{5,1}$

(c) 半边列表

图 3.27　以顶点为索引和以半边为索引的半边模型

半边模型的优点是可以直接记录顶点、边、面片等信息，并且允许在常量时间内执行所有的相邻关系查询。在实际操作中，这样的优势更大，很多情况下，我们的操作都是对附近几何元素的改变，这样就不需要遍历所有的点，实时性强。但是半边模型的数据结构形式相对于简单边界表示更为复杂。

3.7 小 结

本章介绍了计算机图形学中的几何建模技术，包括面向传统 CAD 制造的自由曲线/曲面建模技术、基于图像/视频的三维重建技术以及其他类型的建模技术。不同的技术各有特点，以满足不同应用场合对于形状获取和表示的要求。最后，介绍了面向几何建模的数据结构类型及特点。

虽然这些建模技术能够生成满足要求的形状，但建模的过程仍然需要一定量的人工交互，使得建模的效率较低。由此产生的几何模型的数据规模远不如图像/视频的数据规模。如何通过现有的几何模型有效地生成新的几何形状，也就是数据驱动的几何建模，将是进一步提高几何建模效率的重要途径之一。

思考题

3.1 几何形状的数学表达形式有哪些？各自具有什么特点？

3.2 证明 Bézier 曲线的保凸性。

3.3 B 样条曲线相比 Bézier 曲线的优势有哪些？

3.4 列举具有非矩形拓扑结构的细分曲面，并指出它们在曲面连续性方面的不同。

3.5 从图像恢复三维模型的基本步骤有哪些？

3.6 举例说明分形、粒子系统、语言学方法适用于哪些三维物体的几何建模？

3.7 什么是实体构造几何模型数据结构？它适用于表示哪类三维物体？

3.8 三维物体常用的边界模型数据结构有哪些？各自的优缺点是什么？

第 4 章 数字几何处理

计算机图形学中的几何建模是与数学上的几何学密切相关的。从信息载体的角度来讲，几何和声音、图像、视频等可视媒体数据形式一样，也属于多媒体。随着计算机技术的发展，多媒体先后经历了声音的数字化(20 世纪 70 年代)、图像的数字化(20 世纪 80 年代)、视频的数字化(20 世纪 90 年代)等阶段。到了 20 世纪 90 年代后半期，多媒体开始了几何数字化的阶段。数字几何处理就是对几何建模所得到的三维几何模型通过计算机进行各种方式的处理。

数字几何处理的产生和发展，主要来自于生产生活中对三维模型的广泛需求。例如，Pixar 公司在 1995 年推出了第一部纯计算机制作的电影动画《玩具总动员》，其中的三维动画角色完全是由几何建模所完成。然后，通过对三维模型几何形状的处理产生连续的动画序列，从而进行电影制作。1997 年，Stanford 大学联合 CMU 等其他大学，开展了“数字米开朗基罗”计划，将意大利文艺复兴时期的雕塑进行数字化扫描和建模，将真实的文物转化为几何模型，从而能够进行计算机存储和处理，有力地促进了文化遗产的数字保护。1998 年，时任美国副总统的戈尔提出“数字地球”概念，其核心内容是全球地理信息的数字化。此后，Google、Apple 等公司将地球以三维几何模型的形式在计算机、手机等终端进行展示，为普通大众的日常出行提供了极大的便利。

本章首先介绍与数字几何处理有关的一些几何学基础知识，接下来按照几何模型的数字处理方法，着重介绍网格去噪、网格简化、网格参数化、重新网格化、网格编辑和网格形变等数字几何处理方法。

4.1 几何基础

几何的数字化产生了由离散元素表示的数字几何形状。数字几何处理研究如何基于计算机的存储和计算能力来处理这些数字几何形状。这些形状通过一定形式的几何模型进行数字化表示，进而运用几何学工具进行分析和处理。

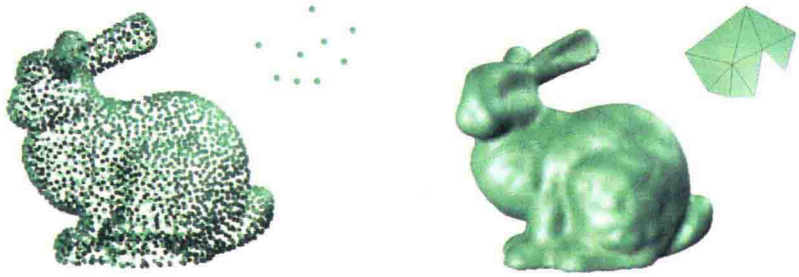
4.1.1 几何模型

几何模型是从数学形状的角度描述物体的外观。除了第 3 章中涉及的几何建模中形状的表示形式外，常用的数字几何模型还包括图 4.1 所示的两种形式：点云和多边形网格。

1. 点云

点云采用离散的三维空间点作为几何形状表示的基本元素，可以记录构成形状的点的三维坐标信息（表示为 $\boldsymbol{v}=(x,y,z)$），以及在该点处的法向信息（表示为 $\boldsymbol{n}=(n_x,n_y,n_z)$）。一般而言，点的数目越多，能够展现出的几何细节越清晰，也就是能描述的几何形状越准确。点云的优点在于数据结构简单，只需逐点记录相应的三维坐标、法向等几何信息，而不用记录不同点之间的连接关系。通过增加点的数目，可以对几何形状进行更准确表示。

然而，点云表示的形状没有任何的表面连续性，而且通常模型的点的数目较多。例如，“数字米开朗基罗”计划中大卫雕塑数字化之后的点云模型包含了 10 亿多个三维空间点，而对应的原始雕塑高 3.96m。此外，由于缺乏连续性，点云数据容易引入形状歧义，很难有效地表示拓扑结构复杂的几何形状。



(a) 点云　　(b) 网格

图 4.1　常用的数字几何模型：点云和网格

2. 多边形网格

用由顶点、边和多边形面组成的集合来表示的几何形状，通常称为网格（Mesh）。例如在图 4.2 中，同样的三维几何模型既可以表示为三角形面片组成的三角网格，也可以表示为四边形面片组成的四边形网格。这类网格可以采用 3.6 节中介绍的边界模型的数据结构进行存储和表示。从数学上讲，多边形网格采用分段线性函数，也就是 C^0 连续函数，逼近三维物体表面的几何形状。显然，使用的多边形数量越多，网格表示的几何形状就越准确和精细。

(a) 几何模型　　(b) 三角网格　　(c) 四边形网格

图 4.2　同一个几何模型分别采用三角网格和四边形网格表示

相比于离散的点云和连续的自由曲面，网格在表示几何形状时具有更大的优势。例如，与 NURBS 相比，网格虽然只具有 C^0 连续性，但它能表示的几何形状不受拓扑限制。

正如第3章中所介绍的，NURBS的定义域必须是四边形。这样就大大限制了其形状表示的能力。与离散的点云相比，网格的多边形顶点之间具有明确的边连接关系。因此，这种网格的表示形式在进行一些几何处理时具有更大的灵活性和可控性。在不同形式的多边形网格中，三角网格由于其形状表示的灵活性而被广泛使用（如图4.2(b)所示）。因此，本章主要针对三角网格介绍相关的数字几何处理方法，但其中的部分方法也适用于四边形网格。

4.1.2 几何性质

网格作为由多边形面片组成的多面体，具有一些数学上的几何性质。在数字几何处理方法中，经常用到以下一些数学上的基本概念和性质。

1. 顶点的度

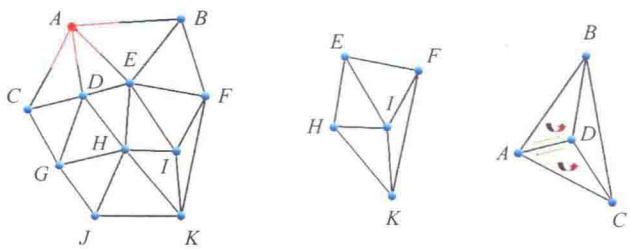
顶点的度是指与一个顶点相连的所有边的数目。顶点 v 的度记为 $\deg(v)$。例如，图4.3(a)中顶点 A 的度为4，与其相邻的边分别是$<A,B>$、$<A,C>$、$<A,D>$和$<A,E>$。其中，度为0的顶点称为孤立顶点。数学上，网格顶点的度和边的数目满足以下形式的握手定律（Handshaking lemma）：

$$\sum_{v\in V}\deg(V) = 2\mid E\mid \tag{4.1}$$

其中，V 是网格所有顶点的集合，E 是所有边的集合，$|E|$表示边的数目。例如，图4.3(a)中所示的三角网格，所有顶点的度数之和为46，而边的数目为23，因而满足公式(4.1)定义的握手定律。

2. 顶点的1-邻域

顶点的1-邻域是指与一个顶点相连的所有三角形面片组成的区域。例如，在图4.3(b)中，顶点 I 的1-邻域就是由顶点 E、F、K、H 所构成。因此，顶点1-邻域内除了该顶点以外的其他顶点的数目就是该顶点的度。



(a) 三角网格　　(b) 1-邻域　　(c) 面定向

图 4.3　三角网格的若干几何性质

3. 边界边

边界边是指有且只有一个相邻多边形面的边。边界边的两个顶点称为边界顶点。具有边界边的网格称为开网格，没有边界边的网格称为闭网格。例如，图4.3(a)所示的三角网格，实际上就是一个开网格。其中，顶点 A、B、C、G、J、K、F 是边界顶点，同时该网格的边界边包括$<A,B>$、$<B,F>$、$<F,K>$、$<K,J>$、$<J,G>$、$<G,C>$和$<C,A>$。

4. 面定向

面定向指构成每一个多边形面的顶点按照顺时针方向或者逆时针方向排列的顺序。如果一条边的两个顶点在与其相邻的两个多边形面定向时的顺序相反，那么这两个多边形面的定向称为相容的(如图 4.3(c)所示)。实际上，多边形面的定向决定了其法向的朝向，以及按什么顶点顺序定义的面是几何模型的正面。

5. 网格拓扑

网格拓扑指组成网格的各种几何元素(点、线、面)的连接关系。拓扑本身是研究连续性问题的数学分支，也就是在连续变换下不变的几何性质。由于网格是 C^0 连续的曲面，网格拓扑也就反映了顶点、边和多边形面之间的连接关系。

6. 连通分支

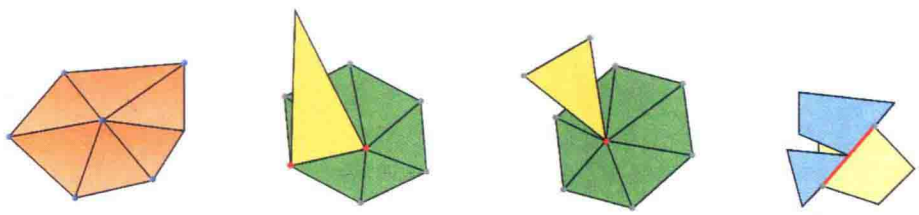
连通分支是由网格上彼此能够通过边形成的路径进行连接的顶点所构成的集合。连通分支的数目是反映网格拓扑的一个重要性质。其中，只有一个连通分支的网格称为单连通网格。这种网格上任意两个顶点之间都可以找到一条由边组成的路径连接起来。

7. 网格亏格

网格亏格指影响网格表面连续性的分割线的数目。亏格是代数几何和代数拓扑中最基本的概念之一。具体是指，若曲面中最多可画出 g 条闭合曲线且不将该曲面分裂，则称该曲面亏格为 g。以闭曲面为例，亏格 g 就是曲面上洞眼的个数。例如，球面没有洞，故 $g=0$；环面有一个洞，故 $g=1$。网格可以看作 C^0 连续的曲面，也具有相同的亏格定义。

8. 流形网格

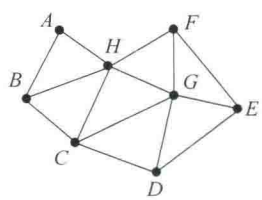
流形网格指局部与圆盘同胚的网格。同胚是指两个集合之间连续的双射。因此，如图 4.4(a)所示，流形网格的每条边只连接一个或者两个多边形面。局部与圆盘不同胚的网格称为非流形网格，典型的是具有呈伞状结构的网格(如图 4.4(b)所示)。可定向的流形网格是其所有面都满足定向相容的流形网格，也就是所有多边形面的定向都是一致的。



(a) 流形网格　　(b) 非流形网格

图 4.4　流形网格和非流形网格

流形网格是一类具有良好几何性质的网格。这类网格满足欧拉-庞加莱特性，也就是说给定一个单连通的闭流形网格 $M=\{V,E,F\}$，它的顶点、边和多边形面的数目满足等式 $g=|V|-|E|+|F|$，其中 g 是网格亏格。



例题 4-1　平面三角网格的几何性质。

问题：写出右图所示的平面三角网格中每个顶点的度、1-邻域顶点以及所有的边

界边。

解答：该三角网格中每个顶点的度和1-邻域顶点如下表所示：

顶点	A	B	C	D
度	2	3	4	3
1-邻域	BH	ACH	$BDGH$	CEG
顶点	E	F	G	H
度	3	3	5	5
1-邻域	DFG	EGH	$CDEFH$	$ABCGF$

进一步，该三角网格的边界边共有7条，分别是$<A,B>$、$<B,C>$、$<C,D>$、$<D,E>$、$<E,F>$、$<F,H>$、$<H,A>$。

□

4.2 网格去噪

通过三维扫描或者图像重建获取真实物体的三维离散点云模型，进而转化为连续的网格模型，这已经成为几何建模的重要途径。但由于采集环境、设备精度等因素的影响，重建的网格表面往往包含不同类型的噪声。在对模型进行几何处理时，首先需要对噪声进行过滤，以此生成表面相对光滑的网格。这个过程就是网格去噪。简单地讲，网格去噪是要消除三维网格模型表面的几何噪声，获得一个表面光滑的几何模型。网格去噪又称为网格平滑、网格滤波等。

4.2.1 基本方法

一般来讲，网格噪声缺乏严格的数学定义，很难区分模型表面出现的是噪声，还是模型本身的形状。这就导致噪声检测也非常困难。从信号处理的角度来讲，噪声是某种信号的高频部分。因此，网格去噪是在消除高频噪声的同时，尽可能地保留几何模型的局部细节特征。这样才能使去除噪声后的网格仍然具有物体原有的几何形状。

形状演化是网格去噪的基本方法。形状演化通过移动网格顶点达到表现平滑状态。这个过程可以描述为顶点位置随时间的变化，即$\partial \boldsymbol{p}/\partial t = F(\boldsymbol{p})$，其中$F$泛指以网格顶点位置坐标作为变量的演化函数。这种方法能够保持网格拓扑，因此不会改变顶点之间的边连接关系。形状演化的过程可以描述为不断地使用网格顶点新的几何位置代替旧的几何位置，最终达到平衡状态，从而生成表面平滑的网格。

形状演化可以采用两种策略：直接的形状演化和间接的形状演化。直接的形状演化，可以表示为$\boldsymbol{p}_{n+1} = \boldsymbol{p}_n + \lambda \cdot F(\boldsymbol{p}_n)$，每次迭代时要用新的网格顶点位置$\boldsymbol{p}_{n+1}$直接取代之前的旧的顶点位置。这也意味着平滑之后的网格顶点可以由原始网格顶点来显式地表示。间接的形状演化，可以表示为$\boldsymbol{p}_{n+1} = \boldsymbol{p}_n + \lambda \cdot F(\boldsymbol{p}_{n+1})$。因此，平滑之后的网格顶点位置$\boldsymbol{p}_{n+1}$需要通过求解相应的函数来获得。这里重点介绍基于Laplacian算子描述几何特

征的网格去噪方法，既有属于直接形状演化的显式 Laplacian 平滑和加权 Laplacian 平滑，也有属于间接形状演化的全局 Laplacian 平滑。

4.2.2 Laplacian 平滑

Laplacian平滑的基本思想是采用 Laplacian 微分算子描述网格局部的几何细节特征。一般情况下，在顶点 $\boldsymbol{v}_i$ 处的 Laplacian 算子定义为

$$\delta_i = \frac{1}{d_i}\sum_{j\in N(i)}(\boldsymbol{v}_i - \boldsymbol{v}_j) \tag{4.2}$$

其中，d_i 是顶点 $\boldsymbol{v}_i$ 的度，$N(i)$ 是顶点 $\boldsymbol{v}_i$ 的 1- 邻域顶点的数目。从微积分的角度来看，公式(4.2) 可以看作曲面上曲线积分的离散化(如图 4.5 所示)。该曲线积分表示为 $\frac{1}{|\gamma|}\int_{v\in\gamma}(\boldsymbol{v}_i - \boldsymbol{v})\mathrm{d}l(\boldsymbol{v})$，其中 γ 是 $\boldsymbol{v}_i$ 点附近的一条封闭曲线，$l(\boldsymbol{v})$ 表示沿曲线 γ 做积分。

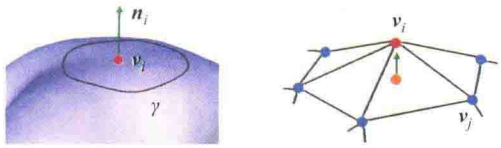


图 4.5 Laplacian 微分算子示意图

当上述曲线长度趋向于无穷小时，就可以得到如下极限情况下的表达式：

$$\lim_{|\gamma|\to 0}\frac{1}{|\gamma|}\int_{v\in\gamma}(\boldsymbol{v}_i - \boldsymbol{v})\mathrm{d}l(\boldsymbol{v}) = -H(\boldsymbol{v}_i)\boldsymbol{n}_i \tag{4.3}$$

其中，$H(\boldsymbol{v}_i)$代表点 $\boldsymbol{v}_i$ 处的平均曲率，$\boldsymbol{n}_i$ 是该点处的单位法向。由公式(4.3)可以看出，Laplacian 平滑是通过 Laplacian 算子计算的向量来描述与平均曲率相关的局部特征。因此，Laplacian 平滑既可以得到相对光滑的表面，又能够在一定程度上保持网格的几何形状。

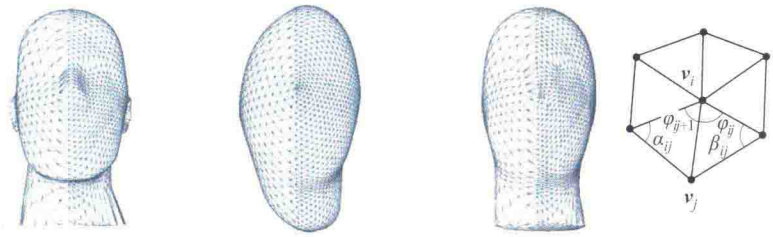
1. 直接 Laplacian 平滑

直接 Laplacian 平滑的基本原理是将每个顶点都移动到 1- 邻域顶点的几何平均位置。这种情况下，在公式(4.2) 中令 $\delta_i = 0$，就可以得到理想的顶点位置，也就是满足 $\boldsymbol{v}_i = \frac{1}{d_i}\sum_{j\in N(i)}\boldsymbol{v}_j$。该过程对每一个网格顶点循环执行，直到相邻两次的顶点误差小于给定的阈值，或者迭代次数达到最大值。直接 Laplacian 平滑算法简单，但邻域内不同的网格顶点 $\boldsymbol{v}_j$ 都使用相同的权重$\frac{1}{d_i}$(称为伞形权重)。因此，该方法不能反映网格局部的几何细节特征差异，导致直接 Laplacian 平滑的结果也无法很好地表示网格的形状(如图 4.6(b) 所示)。

2. 加权 Laplacian 平滑

加权 Laplacian 平滑的基本原理是通过权重来调节 Laplacian 算子对不同网格顶点的作用，从而保持网格的局部几何形状。常用的方法是采用均值权重取代直接 Laplacian 平滑时使用的伞形权重，也就是通过和顶点 $\boldsymbol{v}_i$ 相邻三角形内角的三角函数来定义顶点 $\boldsymbol{v}_j$ 的权重。这种权重的具体形式定义为 $w_j = (\tan\varphi_{ij} + \tan\varphi_{ij+1})/2$，其中 φ_{ij} 和 φ_{ij+1} 是和边

$\boldsymbol{v}_i\boldsymbol{v}_j$ 相邻的两个三角形内角。该权重对应于求解网格上的调和方程所得到的调和系数。此外，典型的权重还有余切权重，具体定义为 $w_k=\cot\alpha_{ij}+\cot\beta_{ij}$，其中 α_{ij} 和 β_{ij} 是和边 $\boldsymbol{v}_i\boldsymbol{v}_j$ 相对的两个三角形内角(如图 4.6(c)所示)。通过加权 Laplacian 算子，可以在网格平滑时更好地保持局部几何形状。



(a) 原始网格　　(b) 直接Laplacian平滑　　(c) 加权Laplacian平滑

图 4.6　直接策略的两种 Laplacian 平滑方法

3. 全局 Laplacian 平滑

全局 Laplacian 平滑属于间接策略。它的基本原理是通过增加约束条件来求解 Laplacian 算子构成的线性方程组。如果采用 $L(\cdot)$ 作为 Laplacian 算子的符号，那么公式(4.2)可以写为：

$$L(\boldsymbol{v}_i)=\boldsymbol{v}_i-\sum_{j\in N(i)}w_{ij}\boldsymbol{v}_j \tag{4.4}$$

这里 w_{ij} 可以设置为相应的伞形权重、均值权重或余切权重。借助于直接 Laplacian 的思想，也就是令 $\delta_i=0$，可以将所有顶点对应的 Laplacian 算子转换为关于顶点坐标的线性方程组，记为 $\boldsymbol{L}\cdot\boldsymbol{V}=\boldsymbol{0}$。该线性方程组也称为 Laplacian 方程组。其中，$\boldsymbol{L}$ 是 $|V|\times|V|$ 的矩阵，$|V|$ 是顶点个数。这里，$\boldsymbol{V}$ 是分别对应 x、y 和 z 三个坐标分量构成的列向量。与直接 Laplacian 平滑方法不同，全局方法将去噪看作网格上所有顶点优化的过程。通过求解 Laplacian 方程组，可以获得平滑后的网格顶点位置。

然而，从上述表达式可以看出，Laplacian 方程组本身是一个齐次方程组。这意味着该方程组存在平凡的零解，也就是使得平滑后的网格顶点退化为同一个点。这显然不能满足网格平滑时保持形状的要求。数学上，Laplacian 方程组的系数矩阵的秩为 $|V|-k$，其中 k 为网格的连通分支的个数。因此，对于单连通的网格，$k=1$，使得方程组系数矩阵的秩为总的顶点数减 1。这就需要至少增加一个约束条件才有可能计算出方程组对应的非平凡解。

通常情况下，选取网格上的 m 个顶点作为控制顶点，使得平滑后的网格形状尽量不偏离这些控制顶点所定义的形状。这些控制顶点的集合记为 $\boldsymbol{C}=\{\boldsymbol{c}_i=(x_i,y_i,z_i)\}_{i=1}^{m}$，其中 $\boldsymbol{c}_i$ 表示第 i 个控制顶点。相应地将 $\boldsymbol{v}_i=\mu\cdot\boldsymbol{c}_i$ 作为约束条件加入 Laplacian 方程组，这里 μ 是约束因子，决定该约束条件对控制顶点的影响程度，也就是网格平滑后偏离网格初始形状的程度。该方程组可以利用线性最小二乘优化，计算唯一的非平凡解，从而得到平滑后的网格顶点坐标。图 4.7 展示了全局 Laplacian 平滑的两个例子。可以看出，通过增加约束的全局 Laplacian 平滑既能够有效去除噪声，也能够很好地保持网格的局部几何形状。

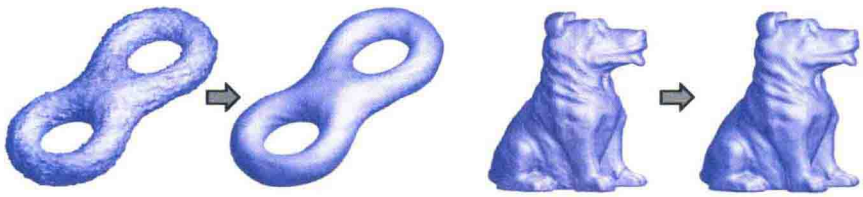


图 4.7 三角网格的全局 Laplacian 平滑

例题 4-2 三角网格的直接 Laplacian 平滑。

问题：假设三角网格 M 具有 n 个顶点，记为 $V=\{\boldsymbol{v}_i=(x_i,y_i,z_i)\}_{i=1}^{n}$。写出通过直接 Laplacian 平滑方法去除网格噪声的伪代码。

解答：假设最大迭代次数为 MAX_NUM，那么直接 Laplacian 平滑时的伪代码是

```
function directLaplacian (M)
    num ← 0
    repeat
        for each vertex v_i in V do          /* 计算每个顶点邻域的几何平均位置 */
            v_i ← Σ_{j∈N(i)} v_j / d_i       /* 顶点 v_i 的邻域 N(i) 有 d_i 个顶点 */
        end for
        num ← num + 1
    until num ==MAX_NUM end
end function
```

□

4.3 网格简化

通过三维重建获得的网格模型通常规模较大，体现在网格顶点和多边形面的数量较多，从而不利于网格模型的存储、传输和显示。网格简化是指通过删除或者修改对几何形状影响不大的几何元素（顶点、边和多边形面等），以减少网格模型的数据规模（如图 4.8 所示）。

图 4.8 网格简化

网格简化时，由于对构成原始网格的几何元素进行了修改，不可避免地造成简化后的网格形状和原始网格形状之间的差异，从而引入形状误差。因此，在进行网格简化时，往

往需要遵循两项基本的原则：顶点最少原则和误差最小原则。顶点最少原则是指在给定误差上界的情况下，使得简化后网格模型的顶点数目最少。误差最小原则，是指给定简化后模型的顶点数目后，简化模型与原始模型之间的几何误差最小。目前，网格简化主要是围绕其中的一项或两项来设计简化算法。典型的方法包括顶点聚类和渐进式抽取。

4.3.1 顶点聚类

顶点聚类是将网格顶点按照其在三维空间中的位置关系进行合并，用合并后得到的较少数目的顶点来代替原始网格顶点，从而实现网格的简化。这种方法主要包含三个步骤：聚类生成、顶点合并、拓扑重构。

1. 聚类生成

聚类生成是根据原始网格相邻顶点几何位置的相近性，将顶点划分成不同的簇。通常情况下，一个簇内的顶点具有空间相近的位置。常用的聚类方法是将原始网格按照其包围盒的空间八叉树进行划分，那么每一个簇对应于空间划分后树节点对应的一个单元。

2. 顶点合并

顶点合并为每一个簇计算一个新的顶点，用于替代这个簇内所有的原始网格的顶点。这样就把原始网格上空间位置相近的顶点合并成一个顶点，进而减少顶点数目。如图 4.9 所示，合并时可以采用不同的策略获得新的顶点。例如，采用簇内顶点坐标的平均值作为合并顶点的位置，这样就用新顶点位置表示了簇内顶点平均的形状；采用簇内顶点坐标的中值作为合并顶点的位置；或者对每个簇内的顶点用二次曲面拟合，然后采用误差最小的顶点作为合并后的顶点，这样就能够在一定误差范围内用新顶点替代簇内原有的顶点。不同策略下得到的顶点合并结果也不相同。通常使用二次曲面拟合的结果要优于平均值或者中值顶点的结果，其原因在于二次多项式具有更高的逼近精度，能够更好地近似原始网格形状。

(a) 平均值　　(b) 中值　　(c) 二次误差

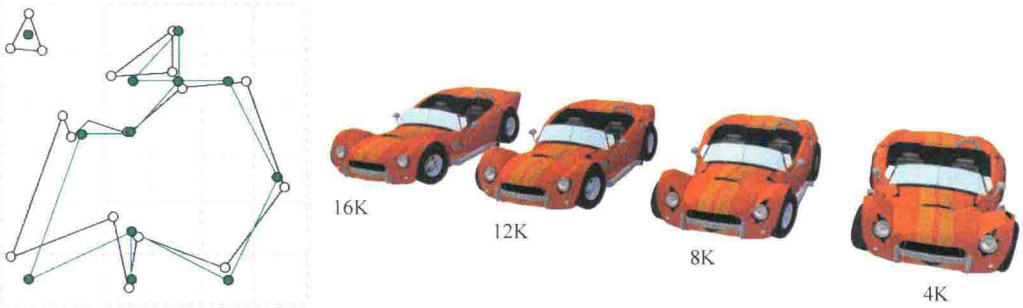
图 4.9　顶点合并时新顶点的生成策略

3. 拓扑重构

拓扑重构是根据合并后新的网格顶点分布，建立顶点之间的边连接关系，生成简化后的网格模型。例如，对于两个聚类后的簇，如果它们在空间中是相邻的，也就是具有相同的边，那么可以在两个簇对应的新的网格顶点之间连接一条网格的边(如图 4.10(a)所示)。

借助顶点聚类的网格简化方法计算速度快，能够用于生成不同规模的简化网格(如图 4.10(b)所示)。但是，这种方法在简化过程中往往难以控制网格拓扑结构的变化，可能会引起简化后网格拓扑连接关系的退化。此外，虽然可以通过二次曲面拟合的新顶点

在全局范围内控制误差，但也是很难生成满足原始网格形状最优的简化结果。

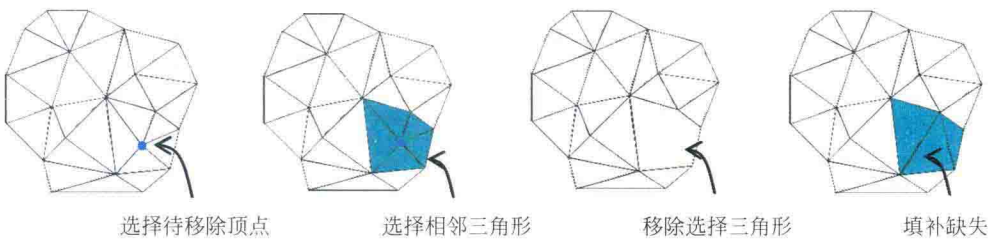


(a) 拓扑重构　　(b) 网格简化结果

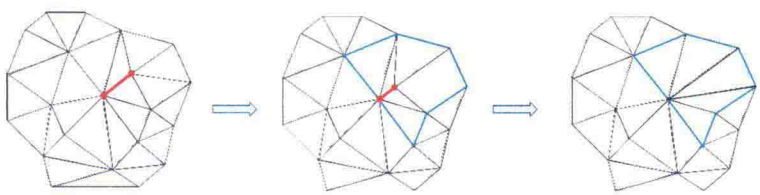
图 4.10　基于顶点聚类的网格简化

4.3.2　渐进式网格

渐进式网格的基本思想是通过预定义的顶点和边的原子操作作为简化算子，然后利用这些算子的组合进行一系列顶点和边的简化，最终实现网格模型数据规模的化简。如图 4.11 所示，常用的原子操作包括顶点移除(DeleteV)和边合并(MergeE)两种简化算子。顶点移除是每次删除网格中的一个顶点，然后对该顶点的 1-邻域组成的多边形进行剖分，从而形成新的多边形面。边合并是在不影响网格拓扑有效性的前提下，将该边的两个顶点进行合并，从而将两个顶点合并为一个顶点，同时该边消失。在合并时，要避免出现顶点翻转现象，也就是避免不同边的交叉的情况，否则会破坏网格的流形性质。



(a) 顶点移除



(b) 边合并

图 4.11　渐进式网格的简化算子

基于上述简化算子，就可以依次对原网格的顶点和边进行简化操作，减少顶点和边的

数目。在简化过程中，需要记录网格原顶点和新顶点位置，以及顶点间的连接关系的变动信息。最后，生成原始网格模型的最简化模型，以及一系列由简化算子构成的渐进序列表示，记为{DeleteV$_0$，MergeE$_0$，DeleteV$_1$，MergeE$_1$，…}。这一系列简化算子所产生的中间网格序列就构成了渐进式网格。

因此，渐进式网格的生成是一个迭代的过程。而在每一次迭代中，往往根据一定的标准选取操作对象。例如，采用简单的随机选择的策略，挑选任意网格顶点或边执行相应的顶点移除或边合并操作。这样每次可以减少一定数量的顶点或边，进而达到模型简化的目的。对于更优的简化方式，则可以根据网格顶点到视点距离远近或者网格表面曲率的大小，采用相应的简化算子，直到顶点或边没有更多减少的可能。相比于顶点聚类的网格简化，渐进式网格能够在网格质量与计算效率之间达到更好的平衡。此外在生成过程中，还可以保证网格拓扑结构不改变。

例题 4-3　三角网格的渐进式网格简化。

问题：假设三角网格 M 包含 n 个顶点和 m 条边。其中，该网格所有边的集合记为 $E=\{\boldsymbol{e}_j=<\boldsymbol{v}_{j1},\boldsymbol{v}_{j2}>\}_{j=1}^{m}$。写出采用随机选择策略进行边合并对网格进行渐进式简化的伪代码。

解答：假设简化操作的最多执行次数为 MAX_NUM，那么采用边合并做简化的伪代码是

```
function PMSimplification (M)
    num ← 0
    repeat
        e_s ← selectEdge (E)                    /* 每次选择一条候选边 */
        v_s ← calculateVertex (v_s1, v_s2)      /* 计算新生成的顶点 */
        deleteEdge (e_s)                        /* 删除候选边 */
        num ← num +1
    until num ==MAX_NUM end
end function
```

□

4.4　网格参数化

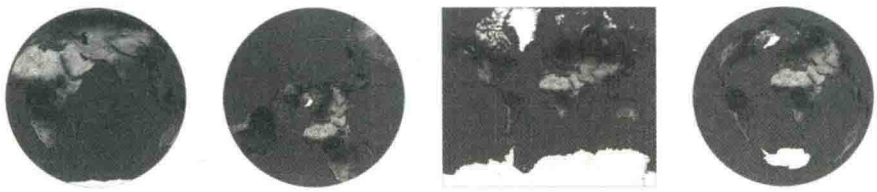
网格参数化是数字几何处理中的基础问题之一。网格参数化最早起源于电影制作时纹理映射的需求。1974 年，Edwin Catmull 首次将纹理映射技术应用于计算机图形学，通过纹理图像和三维模型表面之间的映射关系，计算三维曲面绘制后在屏幕上所对应的每一点的颜色值。这个过程需要建立三维曲面和二维图像之间的一一映射，也就是参数化。曲面参数化后的取值范围称为参数域。

三角网格是常用的三维模型表示形式。三角网格的参数化就是建立网格上每个三角形的顶点与参数域之间的一一映射关系。因此，三角网格参数化的输入是一个三维网格，输出则是一个在参数域上和三维模型同构的，且由三角形组成的平面多边形。

4.4.1 数学模型

地图的绘制可以看作是一种典型的参数化技术的应用，通常可以采用球极坐标的形式将地球表面上的每一点映射到一个理想的球面。但在实际应用中，人们更习惯于将世界地图画在平面图纸上，这样更容易阅读。这样就需要建立三维地球表面和二维平面之间的一一映射关系，称为平面参数化。这种参数化的参数域是二维平面。从世界地图的绘制方式，可以更直观地理解参数化的数学模型。

如图 4.12 所示，典型的地图绘制方法有球极平面投影法、墨卡托投影法和朗伯球面法。球极平面投影是将一个圆球面通过指定极点投射至一个平面的映射。那么除了极点（例如北极）外，该投影能够建立球面上的点到平面上的点之间的双射，而且保持角度不变。这种映射也就是共形映射。但是，球极平面投影时的区域面积会发生改变，尤其在靠近极点附近，会产生局部地区面积的明显变化。墨卡托投影使得投影后的经线变成一系列竖直的、等距离的平行直线，而纬线是垂直于经线的一组平行直线。墨卡托投影也是一种共形映射，也就是在映射时角度保持不变，但面积会发生明显的改变。朗伯等积投影则是一种等面积的平面投影，可以精确记录地区的面积，但是地球表面的陆地形状往往会发生比较大的扭曲。



地球　　球极平面投影　　墨卡托投影　　朗伯投影

图 4.12　地图绘制中的参数化技术

上述三种地图绘制方法所对应的映射可以看作保角度参数化和保面积参数化，描述了参数化映射过程中所保持的各种几何性质。理想的参数化是保形参数化，也就是保持局部形状不发生改变。这里的形状包括角度、面积等几何度量，是对形状的具体描述。保形参数化等价于保持长度不发生变化。但理论上，只有可展曲面存在保形参数化。可展曲面是每一点的高斯曲率都为零的曲面，包括圆柱面、圆锥面、直纹面等少数类型的曲面。一般的曲面在进行参数化时都会存在不同程度的形状扭曲，这是由曲面内蕴几何性质所决定的。因此，对于一般曲线的平面参数化，就是要尽可能减少映射过程中的形状扭曲。

下面以平面参数化为例，构造参数化映射时的数学模型。这里，参数域是二维平面，参数化映射就是建立平面参数域与网格顶点之间的一一映射关系 F，即 $F(u,v)=(x(u,v),y(u,v),z(u,v))$，其中$(u,v)$也称为参数化坐标。在此基础上，参数化映射函数 F 的雅可比矩阵可以写为

$$\boldsymbol{J}(u,v)=\frac{\partial F(u,v)}{\partial(u,v)}=\begin{bmatrix}\partial x/\partial u & \partial x/\partial v\\ \partial y/\partial u & \partial y/\partial u\\ \partial z/\partial u & \partial z/\partial v\end{bmatrix} \tag{4.5}$$

其中，$\boldsymbol{J}(u,v)$是 3×2 的矩阵。在矩阵运算中，雅可比矩阵是函数的一阶偏导数所构成的矩阵，反映了对原始函数的最优的线性逼近。因此，研究雅可比矩阵的性质，对于建立网格的平面参数化映射函数 F 具有重要意义(如图 4.13 所示)。

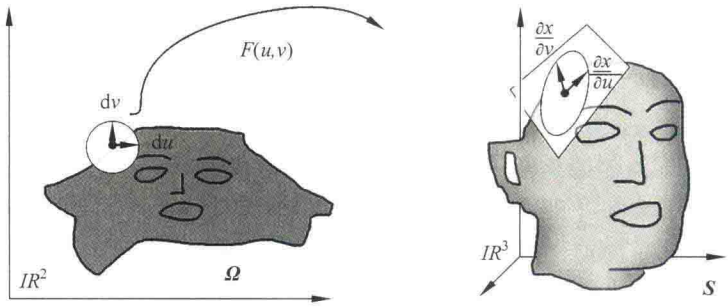


图 4.13 参数化映射函数的雅可比矩阵分析

如果对矩阵 F 进行奇异值分解，可以得到两个奇异值 Γ 和 γ，同时它们满足 $\Gamma \geqslant \gamma$。那么，当且仅当 $\Gamma=\gamma$ 时，F 是保角度参数化；当且仅当 $\Gamma \cdot \gamma=1$ 时，F 是保面积参数化；当且仅当 $\Gamma=\gamma=1$ 时，F 是保形参数化。因此，建立合适的参数化，等价于寻找奇异值满足相应性质的雅可比矩阵，进而反求参数化对应的映射函数，使其雅可比矩阵满足相应的代数性质。

数学上，每一个保形参数化映射都是既保角度又保面积的，反之亦然。此外，按照参数域的不同，网格参数化可以分为平面参数化、球面参数化和基域参数化。平面参数化，是将网格映射到平面，所有网格顶点在映射后分布于平面上。球面参数化，是将网格映射到规则的球面，所有网格顶点在映射后分布于球面上。基域参数化，是采用一个和原模型同构的简化模型作为参数域，所有网格顶点在映射后分布于选定的基域上。其中，平面参数化主要针对开网格，球面参数化和基域参数化主要针对闭网格。下面具体介绍各种不同的参数化方法。

4.4.2 平面参数化

平面参数化选择二维平面作为参数域，建立网格顶点和平面上点之间的一一映射。对于具有边界边的开网格，平面是一个自然的参数域。这是由于两者之间的拓扑同胚性质所决定的。在构建参数化映射函数时，常用的求解模型有弹簧模型、角度展开模型、最小二乘共形模型、几何扭曲优化模型。

1. 弹簧模型

弹簧模型是将网格顶点之间彼此连接的边看作弹簧，那么将顶点位置限定在平面后，在平衡状态下形成的顶点位置分布就是参数化结果。物理上，弹性势能是描述弹簧能量的一种物理量。弹簧系统的弹性势能函数采用如下表达式描述：

$$E=\sum_{i=1}^{n}\sum_{j\in N_i}\frac{1}{2}D_{ij}\,||\,\boldsymbol{t}_i-\boldsymbol{t}_j\,||^2 \tag{4.6}$$

其中，n 是顶点数目，N_i 是顶点 1- 邻域内的顶点数目，$\boldsymbol{t}_i$、$\boldsymbol{t}_j$ 是顶点的空间位置，D_{ij} 是弹簧

系数，描述了弹簧所处的拉伸状态。这里 $\boldsymbol{t}_i$ 可以视作二维或三维空间中的位置。

弹簧系统在平衡状态下满足特定的条件，其势能函数的一阶导数处处为零，也就是顶点分布满足 $\frac{\partial E}{\partial \boldsymbol{t}_i} = \sum_{j \in N_i} D_{ij}(\boldsymbol{t}_i - \boldsymbol{t}_j) = 0$。因此，这就能得到弹簧在平衡状态下的顶点位置满足如下等式关系：

$$\boldsymbol{t}_i - \sum_{j \in N_i} \lambda_{ij} \boldsymbol{t}_j = 0 \tag{4.7}$$

其中，$\lambda_{ij} = D_{ij} / \sum_{k \in N_i} D_{ik}$ 表示相邻顶点的仿射组合系数，构成线性方程组的系数矩阵。那么，如图 4.14 所示，当弹簧顶点限定在平面上时，就得到顶点参数化后在二维平面上的位置，也就是 $\boldsymbol{t}_i = (u_i, v_i)$。

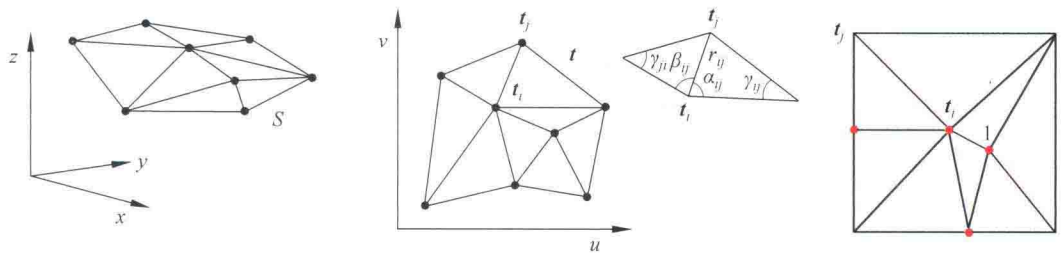


图 4.14　基于弹簧模型的平面网格参数化（图片来自[22]）

公式(4.7)是关于顶点位置的齐次线性方程组。这就需要增加额外的约束条件以计算非平凡的有效解。常用的约束条件是将开网格的边界顶点固定在指定的凸多边形边界上（例如图 4.14 中所示的正方形），也就是将边界顶点在参数化后的平面位置作为约束（例如图 4.14 中的红色顶点）。然后，通过求解线性方程组，就能够得到内部顶点的位置。事实上，这种方式可以看作三角形重心坐标的推广。因为三角形内部每一点都可以通过其三个顶点的线性组合表示，所以类似公式(4.7)定义的参数化映射又称为重心坐标参数化映射。

此外，公式(4.7)中的仿射组合系数 λ_{ij} 也影响了网格参数化结果。根据雅可比矩阵奇异值定义的参数化映射模型，可以将仿射组合系数设置为调和坐标对应的数值。这种情况下的系数具体表示为 $\lambda_{ij} = (\cot\gamma_{ij} + \cot\gamma_{ji})/2$，其中角度 γ_{ij} 和 γ_{ji} 的定义如图 4.14 所示。由于该系数只和角度有关，可以近似地描述从三维空间到二维平面的保角度映射关系。

通过求解基于边界约束条件的弹簧模型，就可以得到有效的平面参数化结果，而且该模型实质上是线性方程组，计算速度快。但是，由于正方形等的凸边界约束，导致参数化结果的几何扭曲较大，尤其是越靠近开网格边界边的位置，几何扭曲的程度越大。

2. 角度展开模型

角度展开模型是按照网格三角形三个内角的分布，重构满足三个内角的平面三角形以建立参数化映射函数。显然，这种方式能够在参数化过程中保持角度不变，因此是一种保角度的参数化。该模型基于三角形内角关系，通过在三角形内角形成的角度空间定义

顶点的参数化映射，利用三角形内角满足的代数关系作为约束，再通过全局优化计算网格顶点参数化后的平面坐标。

在角度空间进行计算时，网格参数化后的顶点构成的三角形应满足四个约束条件：①所有内角为正；②三角形内角之和为 π；③内部顶点周角为 2π；④绕平面顶点一周的三角形封闭。具体来讲，假设第 i 个平面三角形的三个内角是 $\{\alpha_{j=1,2,3}^i\}$，那么根据这四个约束条件可以在角度空间建立如下的目标函数：

$$E(\alpha_j^i) = \sum_{i=1}^{3|T|}\sum_{j=1}^{3}(\alpha_j^i - \beta_j^i)^2$$

其中，α_j^i 满足以下条件：

$$\begin{cases} \alpha_j^i > 0 \\ \alpha_1^i + \alpha_2^i + \alpha_3^i = \pi \\ \sum_{i,k}\alpha_{j(k)}^i = 2\pi \\ \prod_i \sin(\alpha_{j(k)+1}^i) = \prod_i \sin(\alpha_{j(k)-1}^i) \end{cases} \tag{4.8}$$

其中，β_j^i 是网格三角形的内角，$|T|$ 是网格三角形面片的数目，$\alpha_{j(k)}^i$ 表示第 i 个三角形中和顶点 v_k 相邻的内角 $\prod_i$ 表示连乘符号。基于这些角度约束条件，就可以计算有效的平面三角形的内角分布情况。因此，如图 4.15 所示，在选定初始的边之后，就可以按照这些角度依次将三角形在平面上展开，从而得到参数化后的平面网格结果。

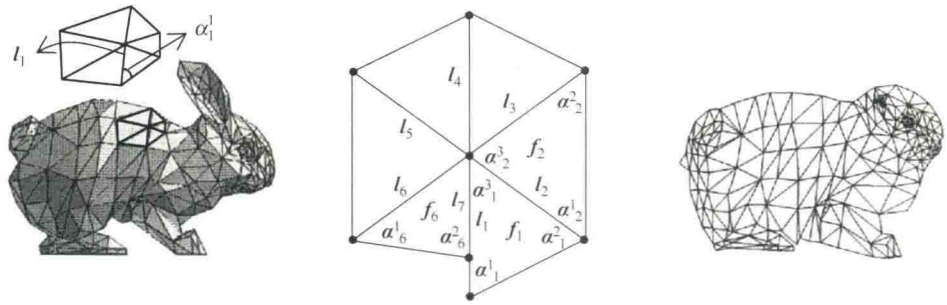


图 4.15 基于角度展开模型的平面网格参数化（图片来自[23]）

基于角度展开的参数化方法，其优点是保证参数化映射后的三角形不会发生边“翻转”，而且整个优化求解的过程收敛。但是，这种方法的缺点是要通过非线性优化（如牛顿迭代法）进行求解，计算效率低，而且参数化后的边界也可能发生交叉。

3. 最小二乘共形模型

最小二乘共形模型是利用几何上的共形映射性质，计算保角度的平面参数化。三维网格的每个三角形可以看作局部的平面三角形。因此，网格的平面参数化就可以转化成三维空间三角形和平面三角形之间的一一映射。数学上，平面上两个形状之间的共形映射 $F:(x,y)\to(u,v)$ 需要满足如下 Cauchy-Riemannian 方程：

$$\begin{cases} \partial v/\partial x = -\partial u/\partial y \\ \partial v/\partial y = -\partial u/\partial x \end{cases} \tag{4.9}$$

其中，$\partial(\cdot)$是计算函数关于其中一个变量的偏导数。进一步，基于三角片的线性特点，上述方程可以写成如下关于参数化后顶点位置 $\boldsymbol{t}_i=(u_i,v_i)$的线性方程：

$$E(\boldsymbol{t}_i)=\sum_{\Delta t_{i1}t_{i2}t_{i3}}|(\boldsymbol{t}_{i1}-\boldsymbol{t}_{i3})-R^{\alpha}(\boldsymbol{t}_{i2}-\boldsymbol{t}_{i3})l_2/l_1|^2 \tag{4.10}$$

其中，$\Delta\boldsymbol{t}_{i1}\boldsymbol{t}_{i2}\boldsymbol{t}_{i3}$对应于三维网格三角形所在的局部平面三角形，$\alpha$ 是边 $\boldsymbol{t}_{i3}\boldsymbol{t}_{i1}$和边 $\boldsymbol{t}_{i3}\boldsymbol{t}_{i2}$的夹角，$R^{\alpha}$ 是 2×2 的旋转矩阵，l_1 和 l_2 是这两条边的长度。

一般来讲，公式(4.10)具有平凡的零解。为了得到有效解，需要额外指定三个点的参数化坐标作为已知条件代入公式，从而将公式(4.10)的求解转化为最小二乘问题进行优化求解。一般情况下，需要指定网格中一个三角形的三个顶点的参数化坐标作为已知条件。这种方法的优点是能够保持参数化过程中的角度不变，而且求解过程是线性的，计算速度快。但是，缺点是无法保证参数化后平面三角形的有效性，有时会发生三角形的“翻转”。

上述三种网格参数化方法是对具有边界的开网格进行处理。如果对闭网格进行平面参数化，通常的做法是首先需要将闭网格切割来产生边界，再运用前面的各种开网格参数化进行计算。因此，闭网格平面参数化的关键是寻找合适的切割线，能够有效地将闭网格切开，变成开网格。常用的切割线计算方法主要有两种：最小生成树和可见性优化。

最小生成树是在网格表面寻找曲率高的网格顶点及边，由其张成的最小生成树作为切割线。可见性优化是希望切割线尽量被隐藏在不容易被看到的顶点位置。这就需要通过分析网格顶点在不同视角下的可见性分布特点，寻找尽可能不可见的边，将其依次连接起来作为切割线。

例题 4-4　三角网格的平面参数化。

问题：输入具有开边界三角网格 M，采用基于弹簧模型的参数化方法进行平面参数化，写出相应的伪代码。

解答：假设三角网格 M 的所有顶点集合为 $V=\{\boldsymbol{v}_i=(x_i,y_i,z_i)\}_{i=1}^{n}$，经过平面参数化后的顶点集合为 $\boldsymbol{P}$，那么平面参数化的伪代码如下。

```
function springParameterization (M)
    vrtOut ← detectBoundary (M)                    /* 检测边界顶点 */
    setBoundary (vrtOut)                           /* 将边界顶点设为平面固定边界位置 */
    L ← 0                                          /* 系数矩阵初始化为零矩阵 */
    for each vertex vi in vrtInt do                /* 计算内部顶点平面位置 */
        Ni ← countNeighbor (vi)                    /* 邻域顶点个数 */
        L[i][i] ← 1                                /* 设置系数矩阵的元素取值 */
        for j ← 0 to Ni do
            L[i][j] ← 1 / Ni
        end for
    end for
    Bx,y,z ← 0                                     /* 零向量 */
    for each vertex vi in vrtOut do                /* 设置边界顶点向量 */
        Bx [index (vi)] ← xi
        By [index (vi)] ← yi
        Bz [index (vi)] ← zi
    end for
```

```
    p ← L^{-1} · B_{x,y,z}                    /* 求解线性方程组得到参数化结果 */
end function
```

□

4.4.3 球面参数化

虽然闭网格可以切割成开网格进行平面参数化，但在切割线附近会导致参数化结果的不连续，具体表现为纹理映射后网格表面图像的跳跃。这本质上是由于闭网格和平面参数域在几何性质上的不同所造成。因此，如何对封闭网格进行整体参数化，是数字几何处理非常重要的问题之一。亏格为零的闭网格与球面同胚，所以对于这类网格，自然的参数域应该是球面，也就是应建立闭网格和球面之间的球面参数化映射，记为 $F:(x,y,z)\rightarrow(u,v,w)$。

球面参数化是将亏格为零的三维封闭网格映射到球面，等价于将网格顶点之间的拓扑连接关系嵌入单位球面。数学上，球面参数化的理论基础是 Steinitz 定理。该定理描述了任何亏格为零且封闭的三角化可以映射成球面三角化。因此，亏格为零的闭网格的球面参数化是一定存在的，这为球面参数化提供了理论基础。

事实上，平面参数化方法中的重心坐标参数化的思想，可以推广到球面重心坐标映射，从而建立网格的球面参数化。假设网格上顶点 $\boldsymbol{v}_i=(x_i,y_i,z_i)$ 在参数化后映射为球面上的顶点为 $\boldsymbol{t}_i=(u_i,v_i,w_i)$，那么球面参数化映射满足以下等式关系：

$$\begin{cases} x_i^2+y_i^2+z_i^2=1 \\ \alpha_i x_i-\sum\limits_{j\in N_i}\lambda_{ij}x_j=0 \\ \alpha_i y_i-\sum\limits_{j\in N_i}\lambda_{ij}y_j=0 \\ \alpha_i z_i-\sum\limits_{j\in N_i}\lambda_{ij}z_j=0 \end{cases} \tag{4.11}$$

其中，$\{\lambda_{ij}\}$ 是顶点 $\boldsymbol{v}_i$ 的 1-邻域的重心坐标（见公式(4.7)中的定义），α_i 是尺度因子。通过上述公式计算得到的顶点一定是位于单位球面上。因此，通过求解该带约束的非线性优化问题，就可以得到网格的球面参数化。直观上，公式(4.11)是将网格每个顶点的邻域顶点的重心坐标投影到单位球面。如图 4.16 所示，G 是 A、B、C、D 四个点的几何重心，那

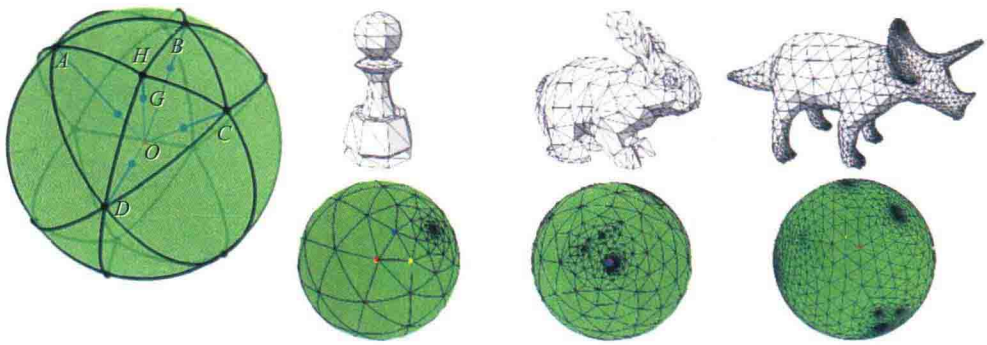


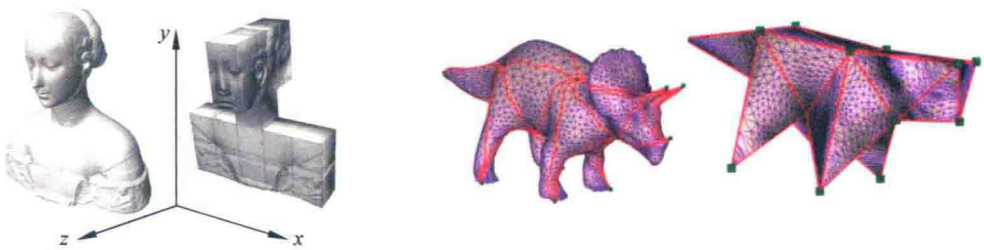
图 4.16 基于重心坐标的球面参数化（图片来自[25]）

么将球心 O 和 G 的连线延长一定倍数，也就是按照尺度因子进行缩放，就可以投影到球面上的 H 点，从而得到球面参数化结果。

4.4.4 基域参数化

当闭网格的亏格大于零时，需要选择与其拓扑同胚的参数域。这类参数化所选择的参数域能够反映任意形状和拓扑的网格，因而将其参数域称为基域，也就是对应于网格本身形状和拓扑性质的一种基本参数域。因为网格参数化需要建立网格顶点和基域之间的一一映射，所以在选择基域时通常采用形状简单、拓扑同胚的基础几何形状，从而方便网格顶点映射为基域上的点。最常用的基域有多面立方体(Polycube)、单纯复形(Simplex)等。它们在形状上能够反映网格的整体外观，在拓扑上具有完全等价的性质。

多面立方体采用和网格形状相近的多面体作为基域。同时，该多面体的多边形面片平行于 x、y 和 z 三个轴平面(如图 4.17(a)所示)。这种定义方式方便了网格顶点到最相近多边形面片的映射。单纯复形则是由三角形按照规定方式形成的几何形状。如图 4.17(b)所示，这些三角形或者不相交，或者相交于公共顶点或公共边。多面立方体和单纯复形在构造时遵循网格顶点的就近原则，因此可以利用投影等直接方法建立网格和这些基域之间的映射关系。同时，为了避免投影扭曲和错位，也会采用后处理的方式来优化投影结果，以使得网格上顶点和基域上投影点之间满足一对一的映射关系。一般来讲，基域参数化可以看作平面参数化和球面参数化的扩展，可用于任意亏格的网格参数化。此外，基域参数化也为后续章节中的网格编辑、网格形变等几何处理提供了基础工具。



(a) 多面立方体　　(b) 单纯复形

图 4.17　复杂几何模型的基域参数化(图片来自[26]和[27])

4.5 重新网格化

同一个几何模型可以表示成不同形式的多边形网格，也就是改变组成模型的每个多边形面片的形状。重新网格化就是把给定的原始网格转换成新的网格表示。例如通过改变原始网格的顶点和边的分布，使新的网格多边形的边长更加均匀。这样就能够使得每个三角形接近于等边三角形的形状。重新网格化的问题来源于三维建模。这是由于直接从点云重建的网格大多具有不规则的多边形，影响了后续网格处理的效率和质量。新的网格在表示原始网格几何形状的同时，能够很好地改善原始网格多边形的质量，以适应于后续实际应用的需要。

4.5.1 网格质量

重新网格化的主要目的是提高原始网格的质量。因此，重新网格化的首要问题是如何定义网格质量，也就是具有什么形状的网格多边形是优质的。显然，网格质量是与使用网格的实际应用场合密切相关的。一般来讲，网格质量是指多边形面片几何形状的规则性，表现为构成多边形的相关几何元素（角度、边长等）的分布属性。

网格质量可以用细长比、锥度比、内角、翘曲度、伸展度、边节点位置偏差等指标来具体量化。例如，细长比是指网格多边形最长边和最短边的比值；翘曲度是指平面在空间中的弯曲程度，如曲率；伸展度则是通过多边形的对角线长度与边长计算出来的，因而仅适用于四边形和六面体单元。

对于三角网格来讲，高质量的网格应该满足三角形的边长近似相等，或者说每个三角形近似于等边三角形（如图 4.18 所示）。这种网格对应的细长比就越接近于 1。下面介绍三种重新网格化的经典方法：渐进式重新网格化、重心 Voronoi 填充重新网格化和基于参数化的重新网格化，其目的都是提高网格质量。

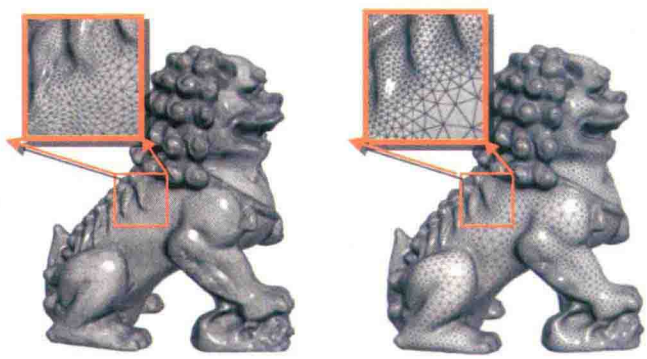


图 4.18 重新网格化改善网格三角形的质量（图片来自[28]）

4.5.2 渐进式重新网格化

这种方法从原始网格开始，利用迭代优化的方法，逐步提高顶点分布的均匀性，同时更新网格边的连接关系，从而生成崭新的高质量网格。渐进式重新网格化主要涉及对顶点和边的操作，因此需要定义相应的操作算子，具体包括点移动、边合并、边分离、边翻转等（如图 4.19 所示）。

点移动，是将每个顶点移动到 1-邻域的几何平均位置，也就是 $\boldsymbol{v}_i \to \sum_{j=1}^{N_i} \boldsymbol{v}_j / N_i$。这样就可以简单地提高局部顶点分布的均匀性。边合并，是将网格上的一条边做压缩，把该边的两个顶点合并为一个顶点。这样可以消除与该边相邻的两个过于狭长的三角形，从而降低边的细长比等指标。边分离，是将狭长三角形的长边划分为相等的两条短边。这样可以将狭长三角形变成边长更加一致的两个小三角形，从而既能够降低狭长比，又可以提高顶点分布均匀性。边翻转，是将较长的边通过翻转，重新连接该边邻近的顶点，得到边长分布更

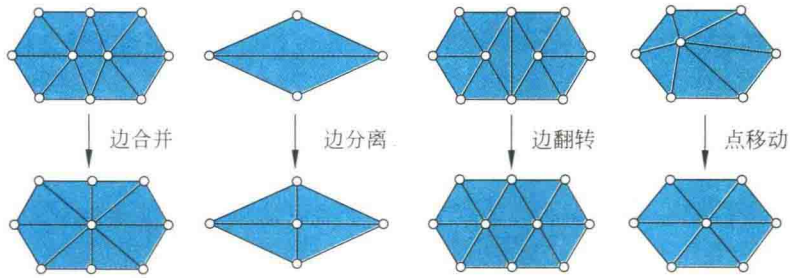


图 4.19 重新网格化过程中对顶点和边的操作

加均匀的网格。边分离和边合并的主要目的是使得三角形边长分布更加均匀，使网格边更接近于一致的边长，使得每个三角形更接近于等边三角形。

渐进式网格优化就是迭代地使用上述几种基本操作，不断地改善网格质量。假设给定重新网格化后的边长具有上限 $l_{\max}$ 和下限 $l_{\min}$，那么目标网格边长 l 应满足 $l_{\min} \leqslant l \leqslant l_{\max}$。在每一步迭代优化时，首先进行边分离、边合并、边翻转等和边相关的一系列操作。具体来讲，首先将网格中所有边长大于 $l_{\max}$ 的边进行分解，通过边分离在其中点处分离成两条边，并和相邻顶点组成新的三角形；然后将网格中所有边长小于 $l_{\min}$ 的边进行合并，从而把该边的两个顶点变为一个顶点。在此过程中，如果可以通过翻转边的连接顶点降低细长比，则进行相应的边翻转操作。这样也可以平衡顶点的度，使其趋近于 6。然后，通过点移动将顶点移动到 1-邻域的重心位置，提高顶点分布的均匀性。最后，将所有顶点投影到网格表面上，得到具有更优质量的网格。

上述迭代步骤重复执行。当所有网格边长满足上、下限条件时，迭代终止，最后生成新的网格。该网格顶点和边的分布更加均匀，使得每个三角形边长尽可能等距，也就是对应于等边三角形，因此可以使重新网格化的结果具有更高的网格质量。

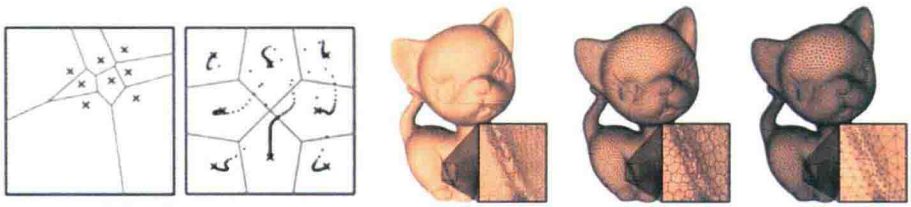
4.5.3 重心 Voronoi 图填充重新网格化

平面 Voronoi 图是由平面区域中连接两邻点的线段的中垂线所形成的区域。重心 Voronoi 图则是一种特殊的 Voronoi 图，其特点是每个 Voronoi 单元对应的采样点正好是区域的几何重心。因此，相比于普通的 Voronoi 图对应的 Delaunay 三角化，重心 Voronoi 图对应的 Delaunay 三角化结果在形状方面更加规整，能够提供更高质量的三角形。重心 Voronoi 图填充重新网格化就是利用重心 Voronoi 图的上述性质，对网格顶点的分布进行优化，提高几何规则性，进而生成高质量的网格。

以平面网格的重新网格化为例，通过构造重心 Voronoi 图以及对应的 Delaunay 三角化可以有效提高原始网格顶点分布的均匀性。因此，这种重新网格化方法的关键是构造重心 Voronoi 图，所采用的基本方法是 Lloyd 迭代。通过该迭代过程，不断地更新 Voronoi 图重心，使其趋近于新生成 Voronoi 单元的重心位置(如图 4.20(a)所示)。假设重新网格化后的顶点个数为 N_m，首先随机地指定其初始位置，记为 $\{\boldsymbol{v}_i\}_{i=1}^{N_m}$。然后，根据顶点位置构造普通的 Voronoi 图。这样，每个顶点 $\boldsymbol{v}_i$ 生成一个划分的 Voronoi 区域 Ω_i。接着，移动 $\boldsymbol{v}_i$ 到 Ω_i 的重心 $\boldsymbol{c}_i$ 处，从而得到更新后的顶点位置。不断迭代上述过程，直到收

敛到稳定状态，也就是重心位置不再发生明显的移动。此时得到的 Voronoi 图就是重心 Voronoi 图。最后，通过重心 Voronoi 图构造对偶的 Delaunay 三角化，这样就可以得到重新网格化的平面三角网格。

上述针对平面网格的重心 Voronoi 图填充重新网格化，可以直接推广到三维空间中的三角网格的重新网格化。利用三维空间中的重心 Voronoi 图可以方便地构造指定顶点个数的新的三角网格（如图 4.20(b)所示）。同样，由于重心 Voronoi 图及其对偶的 Delaunay 三角化在形状表示时的优质特性，采用这种方式进行重新网格化往往可以得到高质量的网格。



(a) Lloyd 迭代生成重心 Voronoi 图　　(b) 重新网格化结果

图 4.20　利用重心 Voronoi 图对三角网格进行重新网格化（图片来自[29]）

4.5.4　基于参数化的重新网格化

相比于直接在三维网格表面进行重新网格化，对平面、球面等简单形状网格做重新网格化相对简单。这是由于平面或球面参数域的几何形状和拓扑简单，而且往往具有显式的参数坐标表示。因此，利用平面或球面进行采样和重新网格化也是提高网格单元质量的很好的思路。这种思路首先利用网格参数化，将三维网格映射到相应的平面或球面参数域，然后在参数域上对网格顶点做重新网格化，再利用参数化映射的逆映射返回三维网格。这样就可以得到原始网格的重新网格化结果。此外，通过参数化映射过程的长度、角度等几何量的控制，可以保证参数域上的采样点重新映射回三维网格时的良好形状特性。

基于参数化的重新网格化，主要是利用平面参数化在平面域上对重新网格化的顶点进行规则采样；或者是利用球面参数化，在球面上进行顶点的规则采样。在这些方法中，如图 4.21 所示的几何图像(Geometry image)方法是比较经典的一类基于平面参数化的重新网格化方法。几何图像实际上是一种网格平面参数化方法。通过该参数化方法将封

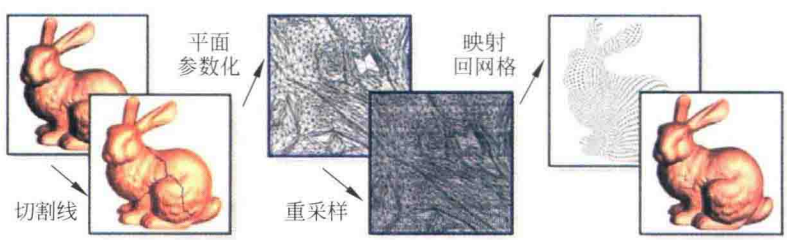


图 4.21　基于平面参数化的重新网格化（图片来自[30]）

闭网格映射到平面参数域；然后在参数域上利用均匀采样计算重新网格化顶点的参数坐标；最后将采样点通过参数化映射的逆映射重新映射回网格表面，得到重新网格化结果。由于这些新的网格顶点对应于平面参数域上的均匀采样，很容易建立相邻顶点的边连接关系，从而生成新的网格。这里需要注意在建立几何图像时，需要预先使用分割线将封闭网格切开成开网格，其本质上是一种开边界网格的平面参数化。

4.6 网格编辑

通过三维重建等方式进行几何建模，需要对每个形状建立相应的几何模型。甚至是同一个物体在不同时刻产生了形状变化，也需要对各个时刻的形状分别建模。这就大大影响了几何建模的效率，特别是在游戏、电影、计算机辅助设计等应用中，这种建模方式费时费力。网格编辑，是操纵和修改现有网格的几何形状的过程，能够实现更有效的几何建模。

网格编辑需要对形状进行修改，是一种交互式的建模方式。因此，网格编辑主要解决的问题有两个：有效的三维模型交互，以及形状改变时的几何细节特征保持。计算机中的三维模型往往通过第三方输入设备（如键盘、鼠标等）进行操作，因此这种交互应当是便捷且尽可能接近现实中对物体的各种操作模式。网格编辑的目标是改变原始网格的形状，但同时需要保留几何细节，使得编辑后的网格仍具有原始网格的外形特征。下面介绍三类典型的网格编辑方法，分别采用了不同的途径实现网格编辑时的交互模式及特征保持。

4.6.1 自由编辑

这类方法是受自由曲面建模方法的启发，通过控制顶点位置的改变来影响几何形状。自由编辑方法的基本思想是将原始网格嵌入到一个简单的控制多面体中。这个多面体称为控制网格。然后通过对控制网格顶点的操控，改变原始网格形状。控制网格通常选取具有比原始网格更简单形状的多面体，这样能够更加容易地进行交互操作。此外，借助控制网格顶点对原始网格形状的约束，可以一定程度地保持编辑后原始形状的几何特征。常用的自由编辑方法包括基于样条的自由编辑和基于重心坐标的自由编辑，它们都是采用控制网格对三维网格进行编辑，但是前者采用样条控制顶点构建控制网格，后者则采用网格简化构建控制网格。

1. 基于样条的自由编辑

基于样条的自由编辑是通过样条函数的控制顶点来定义相应的控制网格。具体来讲，通常采用三变量张量积样条函数来表示原始网格的形状。这样，原始网格就嵌入到由控制顶点形成的控制网格中（如图4.22所示）。从而，网格上的任意一点 $\boldsymbol{P}(s,t,u)$ 可以通过控制顶点的样条函数表达式进行表示，其中 s、t、u 分别对应三个样条函数的参变量。在网格编辑时，通过操纵控制顶点，就可以改变嵌入在控制网格内部的网格表面上点的位置，从而达到编辑形状的目的。由于样条函数的局部连续性，可以实现编辑过程中几何特征的连续变化，从而使得编辑后的网格能够尽可能地保持原始网格的几何特征。例如在

图 4.22 中，虽然拖动一个控制顶点改变了球面部分区域的形状，但在其他不受控制顶点影响的区域，其形状仍近似球面。

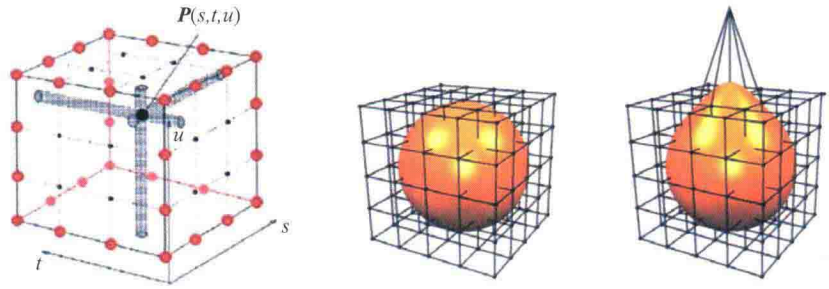


图 4.22 基于样条的自由编辑(图片来自[31])

2. 基于重心坐标的自由编辑

基于重心坐标的自由编辑是将原始网格表面上的点表示为控制网格顶点的线性组合，从而通过改变控制顶点的位置来快速地编辑网格形状。这种方法是三角形重心坐标的直接推广。具体来讲，平面三角形内部的每个点可以通过三个顶点的线性组合表示，那么相应的组合系数称为该点的重心坐标。因此，改变三角形顶点的位置，其内部的每一个点也会发生相应的位置改变，从而影响三角形的形状。类似地，将三角形的重心坐标推广到三角网格表面，那么网格表面上每一点就能够表示为某种形式的控制网格顶点的线性组合。这种组合系数称为三角网格的重心坐标。假设 $\boldsymbol{v}$ 是原始网格表面上的点，$\{\boldsymbol{c}_1,\boldsymbol{c}_2,\boldsymbol{c}_3\}$ 是控制网格上的一个三角形面片的三个顶点，那么相应的重心坐标计算公式为：

$$f_i = \frac{\boldsymbol{n}_i \cdot \boldsymbol{m}}{\boldsymbol{n}_i \cdot (\boldsymbol{c}_i - \boldsymbol{v})}, \quad i = 1,2,3 \tag{4.12}$$

其中 $\boldsymbol{n}_i$ 是锥面的面法向，$\boldsymbol{m}$ 是三角面片法向(如图 4.23 所示)。类似于基于样条的自由编辑方法，通过操纵控制网格顶点的位置，就可以改变原始网格形状。此外，由于重心坐标表示的连续性，这种方法也能够对形状进行连续地改变，从而尽可能地保持局部几何特征。

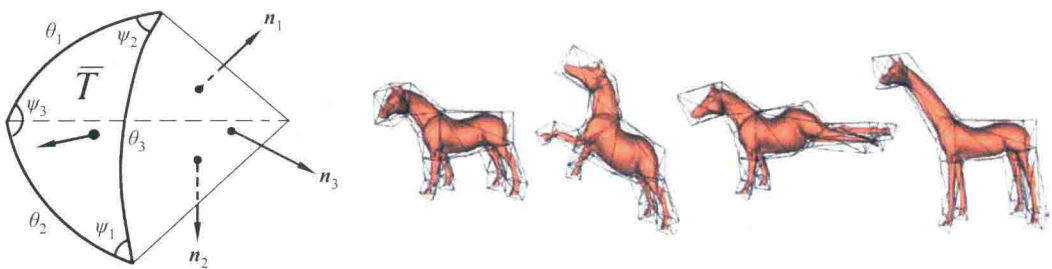


图 4.23 三角网格重心坐标及自由编辑(图片来自[32])

自由编辑的优点是计算简单、快速，能够满足实时形状编辑的应用需求。但其缺点是编辑过程是以几何性质作为约束，难以满足真实的物理效果，也就是说编辑后的形状有可能存在不符合实际形状改变的情况。

4.6.2 带约束的编辑

自由编辑的过程完全依赖于控制网格的改变，编辑效果很难保证物理的真实性和有效性。因此，需要在编辑过程中添加额外约束条件来限制编辑效果。这种方法称为带约束的编辑。常见的方法包括骨架驱动编辑和局部约束编辑，前者借助物体的整体骨架作为约束，后者则是可以施加局部约束来实现更有效的网格编辑。

1. 骨架驱动编辑

骨架驱动编辑主要针对具有肢体的物体，例如动物、人体等。这类物体由于内部存在骨架作为支撑，在编辑时其形状需要根据骨架的分布进行调整。因此，可以考虑先提取代表物体形状的骨架，然后通过改变骨架的形状来驱动外部网格的形状改变，以此实现骨架驱动的编辑效果。此外，通过骨架作为约束，能够一定程度上保持编辑后的物体形状特征。骨架驱动编辑方法多用于计算机动画制作，以方便设计人员更加灵活地控制具有骨架结构的物体变形。

2. 局部约束编辑

局部约束编辑通过对原始网格局部添加约束条件，使得编辑后的形状满足相应的约束限制，同时不影响物体的外观特征。因此，局部约束编辑需要解决两个主要问题：适合交互操纵的局部约束形式，以及保持几何特征的网格变形。局部约束只是对三维网格局部区域内顶点、边、三角形面片几何性质的限制，因此可以通过简单的鼠标单击选取或者笔画勾画来设定约束条件。这些约束条件可以是点/边/面的空间位置、朝向等形式。给定这些约束后，就需要计算满足相应条件的形状，以此得到编辑后的网格。

一种典型的约束形状的方法是 Laplacian 网格编辑方法。该方法借助离散的 Laplacian 微分坐标表示几何细节特征。这样根据约束条件改变顶点位置后，通过计算 Laplacian 微分坐标就可以得到新的网格形状，以此作为编辑后的结果。这里 Laplacian 微分坐标可以看作是 4.2.2 节中 Laplacian 微分算子作用在网格顶点上得到的向量形式。

根据公式(4.4)，顶点 $\boldsymbol{v}_i$ 对应的 Laplacian 微分坐标为 $\delta_i = \boldsymbol{v}_i - \sum_{j \in N(i)} \boldsymbol{v}_j / N_i$，几何上描述了该顶点处的法向朝向和平均曲率(如图 4.24 所示)。借助该微分坐标可以表示局部几何特征，那么编辑后的网格形状在相应顶点处也应该尽可能保持该微分坐标。此外，假设 $\{\boldsymbol{u}_i\}_{i \in C}$ 指定了网格上局部区域内一组顶点在编辑后的位置，那么编辑后的网格顶点 $\boldsymbol{v}'_i \in V'$ 可以通过优化如下形状方程获得：

$$\underset{V'}{\operatorname{argmin}} \left(\| L(V') - \Delta \|^2 + \sum_{i \in C} \| \boldsymbol{v}'_i - \boldsymbol{u}_i \|^2 \right) \tag{4.13}$$

图 4.24 Laplacian 微分坐标及带约束的网格编辑(图片来自[33])

其中 $\Delta=\{\delta_i\}$ 是所有顶点微分坐标的集合。该形状方程使得编辑后的网格形状一方面能够在局部保持原始网格的几何特征，另一方面尽可能满足指定的约束条件 $\{\boldsymbol{u}_i\}_{i\in C}$。

上述方程可以转化为以网格顶点空间位置为未知量的非线性最小二乘优化问题。公式(4.13)中的未知量就是编辑后的网格顶点位置坐标，那么可以通过迭代最小二乘优化进行快速求解，再根据计算的顶点位置获得编辑后的网格形状。

4.6.3 编辑迁移

自由编辑和局部约束编辑都是通过对源网格的直接操纵来达到形状改变的目的。然而，许多物体具有极其相似的几何外形或特征，因此可以将针对一个网格的编辑过程转移到相似的网格上，称为编辑迁移。这种编辑方法就不需要直接去交互编辑源网格形状，而是把已经做过编辑的源网格，映射到需要编辑的网格(也就是目标网格)上，从而得到相应的编辑效果(如图 4.25 所示)。因此，编辑迁移的关键问题是在源网格和目标网格之间建立一一映射关系。这其实也就是网格参数化问题，即将目标网格作为源网格的参数域，寻找符合几何特征的参数化映射。在此基础上，将对源网格的编辑迁移到目标，从而快速地实现对目标网格的形状编辑。

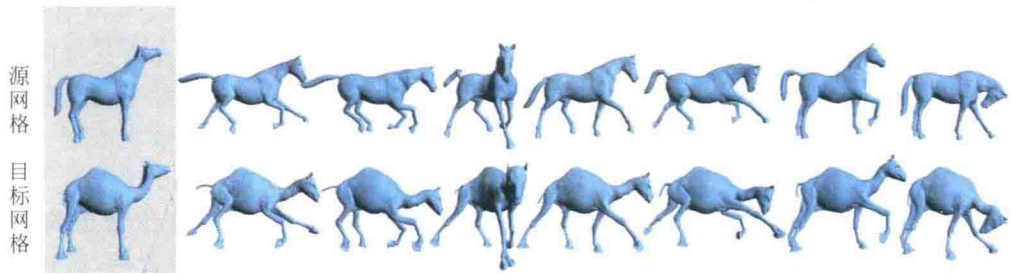


图 4.25 编辑迁移(图片来自[34])

4.7 网格形变

网格形变是利用现有网格形状，实现对新形状的更有效的几何建模。但与网格编辑交互式地改变源网格形状不同，网格形变是在给定源网格和目标网格形状后，通过自动插值方式生成中间过渡形状，从而得到从源网格到目标网格的具有连续形状变化的几何模型。因此，网格形变需要借助于源网格和目标网格之间的映射关系，同时在形变过程中要能够兼顾两者的几何细节特征，以生成合理的中间形状。这些中间形状，就可以作为从源网格到目标网格形变时的网格。

4.7.1 基于参数化的网格形变

给定源网格和目标网格之后，网格形变也可以看作是在两个网格之间建立参数化的过程，也就是将目标网格作为源网格的参数域。因此，通过参数化的方式可以有效建立源网格和目标网格之间的一一映射关系。在此基础上，通过网格对应点之间的插值生成中

间形状。典型的方法是如图 4.26 所示的采用多分辨率自适应参数化(MAPS)的网格形变。该方法包含以下 4 个主要步骤：

(1) 指定源网格 S 和目标网格 T 之间的特征点对和特征线对，从而确保重要几何特征在形变过程中的对齐。

(2) 使用 MAPS 构造基域 S_b 和 T_b，作为源网格和目标网格的参数域。MAPS 借助网格简化思想和边合并的顶点消除法，获得和源网格/目标网格拓扑一致的基域网格。同时，两个基域网格之间的顶点对应形成参数域之间的同胚映射关系。

(3) 根据网格简化过程中顶点的一致性，在基域网格同胚映射关系的基础上，建立源网格和目标网格之间的一一对应关系。

(4) 通过源网格和目标网格对应顶点之间的线性插值等方式，生成中间形状的网格，记为 $M_t = tS_b + (1-t)T_b$。

通过上述过程得到的一系列网格$\{M_t\}$，就可以作为从源网格到目标网格形变时所产生的几何形状。这里需要注意，在第(4)步中可以选择不同的插值方式来生成中间形状的网格顶点位置，而不局限于最简单的顶点之间的线性插值。

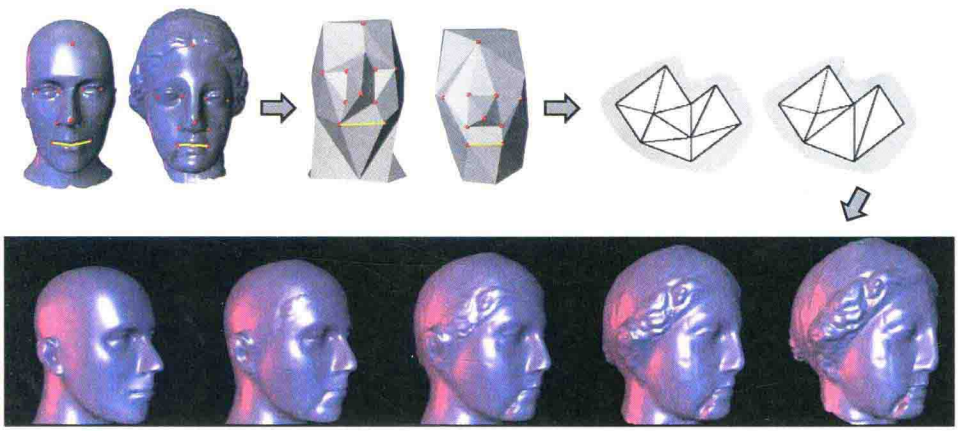


图 4.26 基于 MAPS 的网格形变(图片来自[35])

4.7.2 基于微分坐标的网格形变

通过参数化的方法直接插值源网格和目标网格顶点的三维空间位置，在形变过程中会导致中间形状容易产生体积收缩、几何特征扭曲等问题。这是由于源网格和目标网格之间显著的几何差异所造成的。因此，选择合适的几何特征进行插值，很大程度上影响了网格形变及相应中间网格形状的质量。而利用微分坐标表示网格表面的几何特征，并通过微分坐标的插值实现网格形变，能够更好地保持这些形状的几何特征。

典型的微分坐标形式如 4.6.2 节中的 Laplacian 微分坐标，分别计算源网格和目标网格对应顶点的 Laplacian 微分坐标，记为 $\boldsymbol{\delta}_i^S$ 和 $\boldsymbol{\delta}_i^T$。那么，如图 4.27 所示，中间形状的微分坐标 $\boldsymbol{\delta}_i^{M_t}$ 可以采用 $\boldsymbol{\delta}_i^S$ 到 $\boldsymbol{\delta}_i^T$ 的几何变换的插值来求解。为了使中间形状网格 M_t 能够保持几何特征，该几何变换应该是刚性变换和各向同性的相似变换的复合变换。这样才

能够使得中间形状更好地保持来自于源网格和目标网格的几何特征。

假设从 $\boldsymbol{\delta}_i^S$ 到 $\boldsymbol{\delta}_i^T$ 的几何变换为 $\boldsymbol{H}$，那么通过极分解运算可以将 $\boldsymbol{H}$ 转化为旋转变换 $\boldsymbol{R}$ 和相似变换 $\boldsymbol{S}$ 的乘积，记为 $\boldsymbol{\delta}_i^T=\boldsymbol{H}\cdot\boldsymbol{\delta}_i^S=\boldsymbol{R}\cdot\boldsymbol{S}\cdot\boldsymbol{\delta}_i^S$。这里，$\boldsymbol{H}$ 可以是源网格和目标网格上对应顶点 1-邻域之间的仿射变换。进一步，对分解得到的旋转变换 $\boldsymbol{R}$ 和相似变换 $\boldsymbol{S}$ 分别进行插值，得到作用于中间形状上的几何变换，记为 $\boldsymbol{H}_t=\boldsymbol{R}_t\cdot((1-t)\boldsymbol{I}+t\boldsymbol{S})$，其中 t 代表产生中间形状时的插值参数。这里，$\boldsymbol{R}_t$ 是对旋转变换矩阵 $\boldsymbol{R}$ 对应的旋转角度进行插值后得到的旋转矩阵。那么，将 $\boldsymbol{H}_t$ 作用在源网格的微分坐标上，就得到了插值后的中间形状的顶点所对应的微分坐标。最后，通过求解公式(4.13)对应的形状方程，就可以得到符合微分坐标几何特征的中间形状网格。

相比于直接的顶点插值，基于微分坐标插值的网格形变可以更好地保持形状的局部几何特征，从而得到质量更高的中间形状网格(如图 4.27 所示)。这是由于微分坐标能够比顶点位置更好地反映局部几何特征，从而使插值后的中间形状尽可能地保留了源网格和目标网格的局部形状。

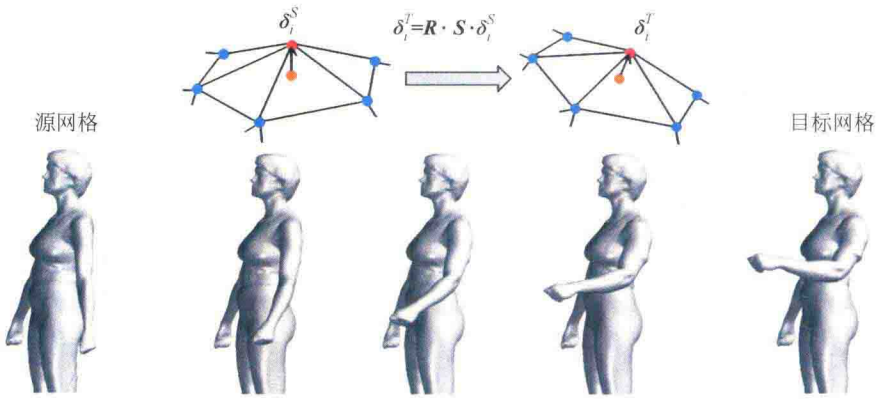


图 4.27 基于微分坐标的网格形变(图片来自[36])

4.8 小 结

数字几何处理是继音频处理、图像处理、视频处理之后的新的数字媒体处理形式。数学上的几何学知识为数字几何处理提供了理论和方法支撑。但是，将连续的几何分析应用于离散的三角网格模型，还是需要根据计算机数据存储和处理的特点，设计网格去噪、网格简化、网格参数化、重新网格化、网格编辑、网格形变等数字几何处理算法。

目前的数字几何处理主要还是从信号处理、函数优化等方面进行算法设计。虽然能够取得一定的处理效果，但在智能化方面还存在很多不足。进一步结合形状的语义分析进行数字几何处理，是当前计算机图形学研究的一个重要问题。

思考题

4.1 点云和多边形网格在表示几何模型时，各自具有什么优缺点？

4.2 流形网格和非流形网格的区别是什么？哪些三角网格是非流形网格？

4.3 三角网格的 Laplacian 平滑算法中，设置不同加权因子对平滑效果的影响是什么？

4.4 渐进式网格简化的简化算子有哪些？

4.5 三角网格参数化的形状扭曲如何度量？

4.6 Voronoi 图和重心 Voronoi 图的区别是什么？

4.7 三角网格编辑时的 Laplacian 微分坐标的定义是什么？数学上有什么含义？

4.8 如何实现两个不同形状的网格之间的形变？

第5章 真实感绘制

计算机图形学中的绘制(Rendering,又称为渲染),是指通过计算机程序将二维或三维几何模型转换为二维光栅图像,从而能够在计算机显示器等终端上显示。计算机图形学中的绘制主要涉及两种类型:真实感绘制和非真实感绘制。真实感绘制是让计算机的绘制结果具有如同相机拍摄真实场景照片般的效果,能够反映出符合自然规律的物体表面颜色、亮度等视觉上可感知的物理属性。非真实感绘制则是不以视觉上的真实感为目标,而是生成具有艺术化效果的图像,以追求特定艺术风格的再现。从计算机图形学诞生后的相当长的一段时间,几何模型及场景的真实感绘制一直是重要问题,直接影响了二维或三维图形的显示效果。

围绕真实感绘制技术,本章着重介绍影响真实感绘制效果的相关因素,如光照、着色等。进一步,根据绘制时所采用的物理模型的不同,重点介绍若干典型的真实感绘制技术,包括纹理映射、光线跟踪方法、辐射度方法等。此外,简单介绍一些特殊效果的绘制技术,包括阴影、毛发等。

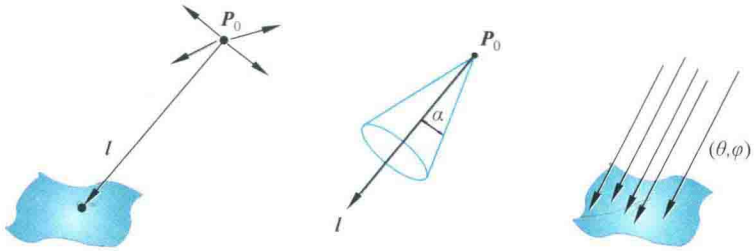
5.1 光　　照

人的眼睛是一个非常复杂的视觉系统。眼睛能够观察到物体外观(颜色、亮度、透明度等),主要是由真实世界中各种光源产生的光照所影响。那些和物体表面作用后进入到眼睛的光线携带了物体表面的属性信息,形成具有不同色彩感受的物体外观。因此,计算机图形学中绘制的真实感可以解释为要让绘制出来的效果,尽可能符合人类所认知的物理规律以及人眼对真实世界观察的结果。

光照主要涉及光源和光线的物理模型。光源,是本身能发光且正在发光的物体(忽略周围环境的反射光),表示了光线的起始位置。光线,则是描述光源发出的光在空间中传播的载体形式。虽然光线在几何上可以看作一条射线,但是其物理上的本质是电磁波,是一种描述了包含不同波长的复色光。人的眼睛能够直接感受到的光线,主要集中在电磁波谱中的可见光范围内(波长范围约为 380～780nm),超出这个范围的光线不会被人眼察觉。光源和光线定义了场景中的光照条件,使得场景中的物体表面呈现出一定的颜色效果。

光源表面上任一点 $\boldsymbol{p}(x,y,z)$所发出的光线通常采用方向和强度来描述。其中,光线方向对应于三维空间向量的方向(θ,φ),光线强度(简称光强)描述了光线传递的能量。这里 θ 和 φ 可以看作球极坐标系中的俯仰角和方位角。那么,由 $\boldsymbol{p}$ 点发出的单条光线可以

表示为$l(x,y,z,\theta,\varphi,\lambda)$，记录了光源位置、方向、波长和强度，其中$\lambda$对应于波长。对于一般的光源而言，其散发出的光强就是沿所有方向上的光线能量的积分。根据发射光线方向的分布不同，光源可以分为点光源、聚光灯、远光等。这些就包含了真实感绘制时场景中光源的主要类型(如图5.1所示)。



(a) 点光源　　(b) 聚光灯　　(c) 远光

图5.1 常见的光源类型

1. 点光源

点光源是一种相比于场景中物体模型的尺寸要小得多的光源。因此，点光源通常可以简化为理想的一个点(如图5.1(a)所示)。同时，点光源向四周均匀发射光线。假设在$\boldsymbol{p}_0$点处的点光源发出的光线强度为$\boldsymbol{I}(\boldsymbol{p}_0)$，那么空间中任意点$\boldsymbol{p}$接收到的光线强度与其到光源的距离成反比，也就是满足$\boldsymbol{I}(\boldsymbol{p})\propto\frac{1}{|\boldsymbol{p}-\boldsymbol{p}_0|}\boldsymbol{I}(\boldsymbol{p}_0)$。因此，离光源越远的位置，其接收到的来自点光源的光强越弱。

2. 聚光灯

聚光灯相比于点光源，其光线方向集中在一个比较窄的范围。点光源发射的光线通常形成圆锥形的半无穷区域(如图5.1(b)所示)。因此，聚光灯可以在点光源基础上，加上一定的角度限制来表示，使其发射的光线聚集在偏离某一个方向的角度范围内。例如将聚光灯表示为顶点在$\boldsymbol{p}_0$、中心轴方向$\boldsymbol{l}$的圆锥，其顶角为α。如果中心轴和母线的夹角$\alpha=180°$，那么聚光灯变为点光源，就会朝空间中各个方向发射光线。此外，点光源的光强分布函数定义为$\boldsymbol{I}(\boldsymbol{p}_0)\cos^e\alpha$，其中指数$e$表示了光强衰减的快慢。因此，在离中心轴越近的地方，受光源发射光线的光强影响越大。

3. 远光

顾名思义就是光源位置位于无穷远处。因而，远光可以认为其发出的光线是平行光(如图5.1(c)所示)。在三维空间中，通常采用球极坐标中的俯仰角θ和方位角φ来指定光线的方向，并以此作为远光模型。因此，远光就是沿着固定方向具有处处均匀光强的模型。

上述各种光源模型都只是对自然界中常见光源的近似模拟。在实际的光照环境下，场景中往往包含不同类型的多个光源。例如，学校教室的场景中除了室外投射进来的太阳光以外，通常也包含日光灯、投影仪光源灯等来自于多个光源的光线。这些光源的位置和光强都会直接影响到该场景绘制的最终效果。

此外，环境光也是模拟真实光照的一个重要组成部分。环境光用于描述空间中处处

均匀的光照效果，使得场景中各处都具有相同的光照强度。严格意义上说，环境光也是来自于某些其他光源，但在光强计算时进行了简化。通常采用从光源发出的光线经过多次反射后的效果来模拟环境光。这样，空间中的光强就是一个常值，与光源位置、方向等因素无关。因此，环境光满足空间均匀分布的条件：

$$I(x,y,z,\theta,\varphi,\lambda) = I_0$$

基于上述光照模型，接下来就可以按照现实世界中光线与物体的作用情况来对计算机中的几何模型和场景进行真实感绘制。真实感绘制技术可以分为两种主要类型：局部绘制技术和全局绘制技术。局部绘制技术，侧重于模拟光线与物体表面的局部作用过程。例如 5.2 节中介绍的 Gouraud 着色模型、Phong 着色模型等。局部绘制技术的优点是算法复杂度低、计算速度快，因此适用于实时性要求高的应用场合，如 3D 计算机游戏。但其缺点是在许多物理场景下，绘制效果的真实感不足，尤其是对包含多个光源的复杂场景的绘制。

全局绘制技术，侧重于光线从光源到成像的整体作用过程，例如 5.4 节中介绍的光线跟踪模型，5.5 节中介绍的辐射度模型。全局绘制技术可以更准确地模拟反射、折射、阴影、色彩扩散等更复杂的光和物体作用效果，绘制效果的真实感强。但其缺点是往往需要复杂的计算过程，整体运行效率比较低。因此，全局绘制技术主要用于电影制作中静态帧画面的逐帧绘制。

从计算机图形学诞生开始，真实感绘制技术就不断革新，涌现出大量不同的局部/全局绘制模型。如图 5.2 所示，在 1967 年 Utah 大学的 Chris Wylie 首先在绘制模型中引入了光照，从而开启了真实感绘制的发展历程。1970 年，美国 UIUC 大学的 Jack Bouknight 提出了第一个光照反射模型，也就是包含朗伯漫反射和环境光的光照模型。在这之后，Utah 大学的研究人员又相继提出了 Gourand 着色模型和 Phong 着色模型等局部绘制技术，使得计算机图形学真实感绘制技术被广泛应用。从 20 世纪 80 年代开始，全局绘制技术开始兴起，包括光线跟踪模型、辐射度模型等。此后，一些对这些全局光照模型进行改进的算法不断被提出，同时也在影视、游戏等行业中得到了广泛应用。接下来，具体介绍这些典型的真实感绘制技术。

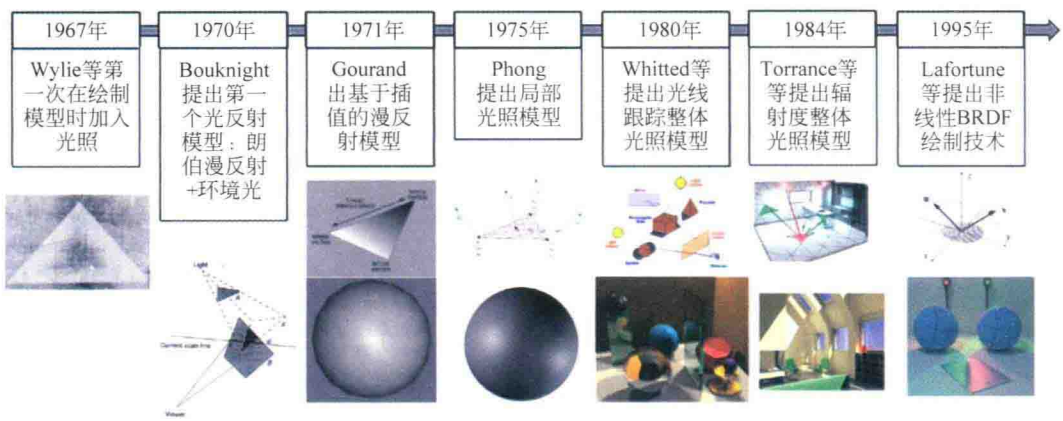


图 5.2 真实感绘制技术发展的简明历程

5.2 BRDF 和着色

除了光照因素外，即使在相同的光照条件下，对同一个物体设置不同的材质属性，也能够绘制出具有不同真实感效果的图像。材质，也就是材料和质感的结合，反映了物体的物质特性。这些特性通常涉及物体表面的色彩、纹理、光滑度、透明度、反射率、折射率等属性。正是物体有了这些不同的物理属性，材质才会影响光线和物体之间的相互作用，进而影响物体最终的绘制效果，如图 5.3 所示。

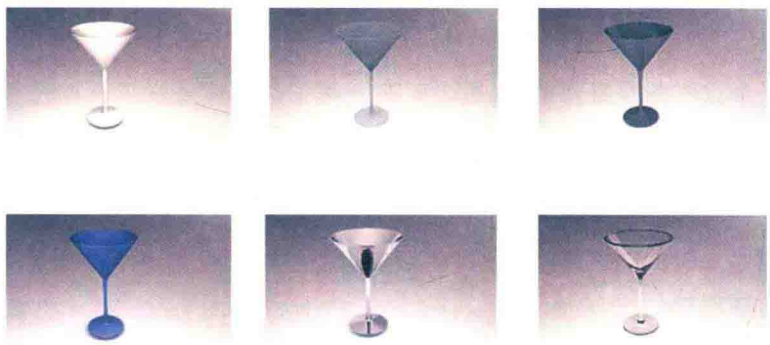
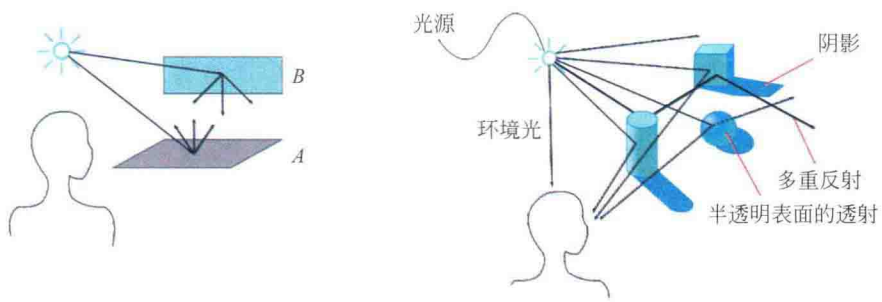


图 5.3 同一物体设置不同材质的绘制效果

一般情况下，光源发射的光线照射在物体表面时，部分光线会被物体吸收，而部分光线会被反射或折射，继续在空间进行传播。从局部来看，如图 5.4(a)所示，光线照射到 A 点，会产生被反射或吸收的光线。一些反射的光线照射到 B，同样会产生被反射或吸收的光线。如果场景中有多个物体，那么上述过程又会在不同物体之间重复循环，直到最终达到一个稳定状态。从整体来看，各种光源发出的光线与场景中的物体发生相互作用。因此，场景的最终绘制效果取决于光线与场景中物体的多次作用程度，如图 5.4(b)所示，并随之产生折射、半透明、阴影、雾化等效果。



(a) 光线与物体的相互作用　　(b) 各种绘制效果

图 5.4 光照与材质影响绘制效果

从物理上讲，光线和物体表面作用时的反射程度、方向等是与物体表面的光滑度、反

射率、法向朝向等因素密切相关的。粗糙的表面会将入射光线朝各个方向均匀散射，称为漫反射(Diffuse Reflection)；光滑的表面则会产生镜面反射(Specular Reflection)，形成亮度高度集中的高光效果，也就是物体表面最亮的局部区域。此外，如果物体本身是透明的，那么有些光线也会透射进物体，形成(半)透明的效果。双向反射分布函数(Bidirectional Reflection Distribution Function，BRDF)是一种描述光线与物体表面作用的物理模型，表示了在物体表面上每一点处将光线从任何一个入射方向反射到任何一个出射方向的反射特性。接下来，首先介绍 BRDF 的相关概念，然后介绍实际绘制模型中所采用的相关技术。

5.2.1 BRDF

想象有一个点光源照射到桌面，那么桌面上每一点都只接收到该点光源的光照。如果从各个不同的方向来观察那个亮点，会发现亮点的亮度随着观察方向的不同而发生了改变。另外，如果视点不动，改变光源和桌面的相对位置，也会发现亮点的亮度发生了相应的改变。这说明在给定物体材质后，一个表面对不同的光线入射角和反射角的组合，拥有不同的反射率。BRDF 就是利用这种表面反射性质对光线和物体表面作用以及传播进行全面描述的物理量。

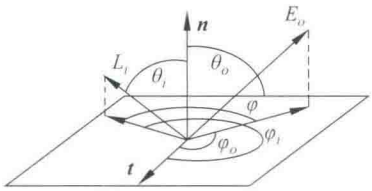
具体来讲，BRDF 是描述光线与物体表面交互的模型，定义为给定出射方向上的辐射能量与入射方向上的辐射能量的比率。因此，BRDF 可以记为如下表达式：

$$f(\theta_i,\varphi_i,\theta_o,\varphi_o,\lambda)=\frac{\mathrm{d}E_o(\theta_o,\varphi_o)}{\mathrm{d}L_i(\theta_i,\varphi_i,\lambda)} \tag{5.1}$$

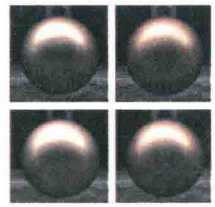
其中，(θ_o,φ_o)和(θ_i,φ_i)是采用球极坐标定义的三维空间中入射光线和出射光线的方向(如图 5.5(a)所示)，$\mathrm{d}L_i$ 和 $\mathrm{d}E_o$ 分别代表入射光能量和出射光能量，λ 则是入射光线的波长。这里，由于一般的 BRDF 需要处理物体表面上半球范围内的各个方向，所以采用了出射或入射方向相对于法向的夹角 θ_o 和 θ_i，称为极角(Polar angle)，以及方向在平面上的投影相对于平面上一个坐标轴的夹角 φ_o 和 φ_i，称为方位角(Azimuthal Angle)。那么，对于各向同性的物体材质，当入射方向和出射方向同时绕法向旋转时，BRDF 值保持不变，此时可以表示为 $f(\theta_i,\theta_o,\varphi,\lambda)$。

从公式(5.1)可以看出，BRDF 的输出结果是一个数值，表示了在给定的入射条件下，在出射方向上反射的能量相对于入射能量的比重。显然，不同的材质会造成物体表面各点处不同的比重值。从物理学光子的概念出发，BRDF 也可以解释为入射光子沿特定方向离开的概率。而从物理学辐射度的概念考虑，由公式(5.1)定义的 BRDF 实际上是辐射率和辐照度的比值。辐射率(Radiance)，表示每单位立体角在单位面积上的辐射通量，也就是单位立体角范围内的辐射度。辐照度表示到达单位面积上的辐射通量。这样就可以简洁地描述出入射光线经过某个表面反射后，如何在各个出射方向上分布。

BRDF 对于模型绘制是非常重要的。因为通过 BRDF，就可以了解光线与不同材质的物体表面的相互作用情况，进而反映了物体本身的物质属性。那么在此基础上，光照模型就会变得非常简单。例如，通过对物体表面的 BRDF 进行积分求和，便可以获得表面上的光强分布，进而得到绘制结果(如图 5.5(b)所示)。



(b) BRDF模型



(b) 绘制效果

图 5.5 基于 BRDF 的绘制效果

利用 BRDF 进行绘制的关键是如何预先获取满足相应材质属性的数据，从而简化绘制时针对 BRDF 数值的实际计算过程。目前主要有三种 BRDF 模型用于数据的获取：经验模型、物理模型和测量模型。

1. 经验模型

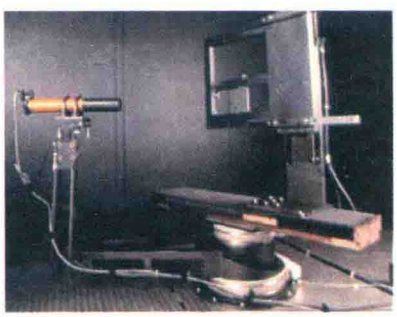
经验模型是根据光与物体相互作用规律的经验提出的数学模型。本节将着重介绍若干典型的经验模型，例如 Gouraud 模型、Phong 模型等。这些模型的绘制方程中所涉及的各类漫反射系数、镜面反射系数、高光系数等都是根据经验设置，并非来自实际的物理规律。在此基础上，计算相应的 BRDF 数值。因此，经验模型不一定符合真实世界的物理规律，但能够模拟大部分光学现象和光照效果，计算量相对较小。

2. 物理模型

物理模型是指根据物理原理对光线传播及其与物体表面的作用进行建模。例如 5.4 节中介绍的辐射度模型，它就是基于热传导模型来模拟光能量在空间中的传播和分布情况，以此获得 BRDF 数值。这种模型符合物理规律，但是计算量比较大。

3. 测量模型

测量模型是指利用光学设备从实际物体的表面直接来获取 BRDF 数据。例如采用测角仪、图像双向反射计，都可以根据入射角和出射角测量得到反射信息，从而得到 BRDF 数值。这种模型可以更为准确地获取实际场景中光照分布情况，但是数据采集的成本较高，往往需要借助专业设备，并且构造专门的数据集进行存储。图 5.6 展示了一种 BRDF 测量设备—光学测角仪，能够捕捉来自不同方向的反射光线。在此基础上，就可以对不同材质的物体表面进行测量，建立 BRDF 数据集。



(a) 光学测角仪

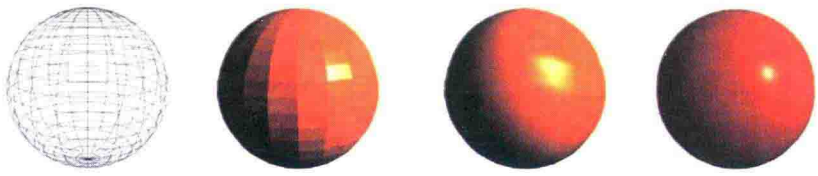
(b) 美国 MERL 实验室 BRDF 数据集

图 5.6 BRDF 数据(图片来自[38])

从计算机图形学发展历史来看，经验模型是最早被提出且广泛使用的 BRDF 模型，解决了许多几何模型和场景在实时真实感绘制时的问题。因此，接下来着重介绍若干基于经验模型的 BRDF 模型，又称为着色(Shading)模型。

5.2.2 着色

着色，顾名思义就是要确定几何模型在屏幕上显示时所对应的像素颜色。本节介绍的着色模型属于局部绘制技术的范畴，也就是只考虑光线和物体表面上每一点处作用所产生的绘制效果。这种着色技术大致可以分为两种类型：平直着色(Flat Shading)和平滑着色(Smooth Shading)。这两类技术主要针对多边形表示的多面体几何模型的绘制。平直着色，是物体表面上属于同一个多边形的所有像素均使用同一种光强效果来填充，例如选用第一个顶点的颜色作为该多边形内部所有点的颜色。因此，平直着色往往产生不连续的视觉效果，表现为如图 5.7(b)所示的相邻多边形在边界处产生明显的差异，称为马赫带效应。平滑着色，是对顶点的颜色、法向等属性进行插值运算，使得多边形面片内部各点的光强效果都会有所变化，由此产生表面连续的视觉效果。Gouraud 着色模型和 Phong 着色模型是两种典型的平滑着色技术(如图 5.7(c)和图 5.7(d)所示)。



(a) 几何模型　(b) 平直着色　(c) Gourand着色　(d) Phong着色

图 5.7　局部着色模型绘制效果对比

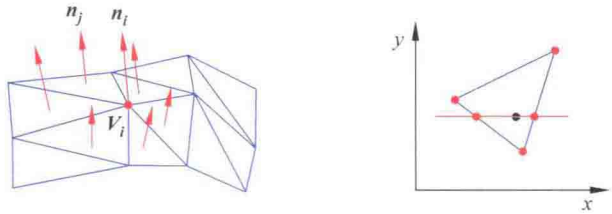
1. Gouraud 着色模型

Gouraud 着色模型是通过插值多边形顶点的光强来产成连续的绘制效果。该模型是由 Utah 大学 Henri Gouraud 在 1971 年提出。从绘制结果的视觉效果来看，Gouraud 着色能够比平直着色产生更加光滑的颜色分布效果，有效减少了物体表面上相邻多边形过渡时的颜色变化，一定程度上减轻了马赫带效应。

Gouraud 着色算法主要包括三个步骤：① 计算每个多边形顶点处法向量；② 采用光照模型计算顶点处光强；③ 采用双线性插值计算多边形内部每点的光强。该算法中每个顶点的法向量，大多是通过计算所有共享该顶点的多边形面片的法向量的平均值得到。这实际上是对该顶点处物体表面弯曲程度的一种近似表示。如图 5.8(a) 所示，顶点 $\boldsymbol{v}_i$ 的法向量记为 $\boldsymbol{n}_i = \sum_{j=1}^{K} \boldsymbol{n}_j / K$，其中 K 表示和该顶点相邻的多边形的个数。接下来计算每个顶点处的光强。这需要根据光源、材质等性质进行计算，相应的内容会在 Phong 着色模型中一并介绍。最后，对顶点的光强进行双线性插值，得到多边形内部每点的光强(如图 5.8(b)所示)。

Gouraud 着色模型计算简单快速。由于是对顶点光强进行线性插值，在边界处的颜色连续性相比平直着色有了较大提高。但是，对于简单的多面体几何模型，仍然存在马赫

带效应，其本质原因在于光强的线性插值，而光强又是根据模型表面的几何形状进行计算。同时，Gouraud 模型难以较准确地描述高光现象，因而不适合具有优良镜面反射特性的光滑物体表面的绘制。

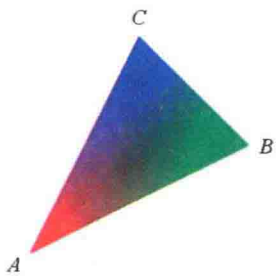


(a) 顶点法向计算　　(b) 顶点颜色插值

图 5.8 Gouraud 着色模型

例题 5-1 三角形 Gouraud 着色的颜色插值。

问题：假设右图所示的平面三角形，其三个顶点 A、B、C 的坐标分别是(0，0)、(2，1)和(1，2)，对应的 RGB 颜色分别是(255，0，0)、(0，255，0)和(0，0，255)，试写出采用 Gouraud 着色的颜色插值绘制该三角形的伪代码。



解答：假设 A、B、C 三个顶点组成的三角形是 f，那么对每一个内部像素(x, y)进行着色的伪代码是

```
function Gouraund (L, M, V, f)
    for each pixel (x,y) in f do
        k1 ← |y -A.y| / |C.y -A.y|
        k2 ← |y -B.y| / |C.y -B.y|
        D ← Point (k1 × C.x - (1 -k1) × A.x, y)
        E ← Point (k2 × C.x - (1 -k2) × B.x, y)
        colorD ← k1 × colorC + (1 -k1) × colorA
        colorE ← k2 × colorC + (1 -k2) × colorB
        k ← |x -D.x| / |D.x -E.x|
        color[x][y] ← k × colorE + (1 -k) × colorD
    end for
end function
```

□

2. Phong 着色模型

为了进一步提高平滑着色的视觉效果，Utah 大学的另一位研究人员、Bui Tuong Phong 改进了之前的 Gouraud 着色模型，实现了对局部光照模型更加准确的模拟。与 Gouraud 着色时插值顶点光强不同，Phong 着色技术首先插值多边形顶点处法向量，然后根据光源位置、插值得到的法向等计算多边形内的每一点处的光强。这样在进行光照计算时，充分考虑了物体表面的几何形状。因而，Phong 模型可以更准确地模拟光照效果。

Phong 着色算法主要包括三个步骤：

(1) 计算多边形顶点处法向量；

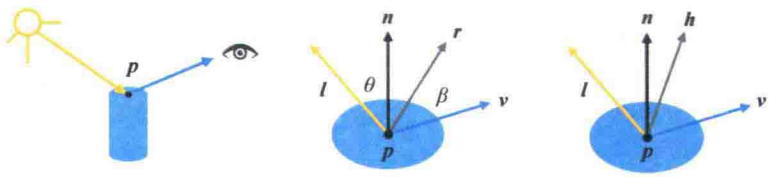
(2) 采用双线性插值计算多边形面片每一点处的法向量；

(3) 通过插值法向量计算每点处的光强。

因此可以看出，Phong 着色和 Gouraud 着色的主要区别在于第(2)步和第(3)步。其中，第(2)步可以通过类似 Gouraud 着色时光强插值的方式对法向量进行插值，也就是对法向量的分量依次进行插值。下面具体介绍第(3)步，这也是 Gouraud 着色模型中计算光强的基本方法。

通常情况下，物体表面上某点处的光强可以看作三种光照分量与物体表面作用后光照效果叠加的结果。这里具体包括三种光照分量，分别是环境光 L_a、漫反射光 L_d 和镜面反射光 L_s 所产生的效果。其中，漫反射光和镜面反射光是点光源发射的光线和物体表面作用的效果。物体表面对这三种光照具有不同的反射属性，分别通过环境光反射系数 k_a、漫反射系数 k_d 和镜面反射系数 k_s 来描述。这三种系数反映了物体材质对不同类型的光线能量吸收和反射的程度。通常情况下，这三种系数取值都是介于 0 到 1 之间。

如图 5.9 所示，假设物体表面上的 $\boldsymbol{p}$ 点到光源的方向矢量是 $\boldsymbol{l}$、到人眼视点的方向矢量是 $\boldsymbol{v}$，同时在 $\boldsymbol{p}$ 点处的法向是 $\boldsymbol{n}$、镜面反射的方向矢量是 $\boldsymbol{r}$，那么上述各种光照效果可以通过以下的方式分别进行计算。



(a) 入射光线与视线　(b) 简单 Phong 模型　(c) Blinn-Phong 模型

图 5.9　Phong 和 Blinn-Phong 着色模型中的光线分布

1) 光照效果 1——环境光反射

由于环境光在空间中每一点处的光强是均匀分布的，那么对应于环境光反射强度可以表示为 $I_a = k_a L_a$。因此，物体在环境光照射下各点明暗程度是一样的。

2) 光照效果 2——漫反射

由于光线照射在物体表面后会朝各个方向均匀反射光线，那么同一点在不同角度下看到的表面就会呈现相同的亮度。这就意味着漫反射光强和反射光线方向无关，而入射光线和法向夹角以及漫反射系数 k_d 就会影响漫反射光强。具体来讲，漫反射光强可以表示为 $I_d = k_d L_d \cos\theta = k_d L_d (\boldsymbol{l} \cdot \boldsymbol{n})$，其中 θ 是表面法向 $\boldsymbol{n}$ 与入射光线方向相反的向量 $\boldsymbol{l}$ 的夹角(如图 5.9(b)所示)。进一步，将前面所述的环境光反射效果加入到漫反射效果，构成了如下朗伯光照模型：

$$I = k_a I_a + k_d I_d (\boldsymbol{l} \cdot \boldsymbol{n}) \tag{5.2}$$

其中，k_a 和 k_d 分别是环境光反射系数和漫反射系数。通过公式(5.2)，就可以准确地模拟各向同性的粗糙物体表面上的光照效果。

3）光照效果3——镜面反射

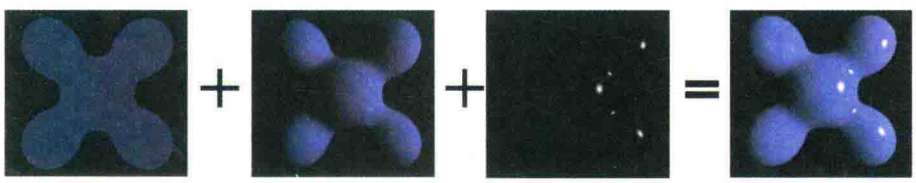
遵循光的反射定律，也就是反射角等于入射角。因此，人眼只能在表面法向的反射方向一侧才能看到入射光的反射光。然而，实际物体表面复杂的物理性质会使得光滑表面呈现出一定的高光区域，也就是亮度集中的区域。这是由于入射光被反射后，绝大多数会集中在理想反射方向的附近区域。高光区域的大小可以通过镜面反射系数 k_s 指定。一般情况下，镜面系数越大，物体表面形成的高光区域越小，因而反射光强与反射光线和法向的夹角以及镜面反射系数有关。具体来讲，镜面反射光强可以写为 $I_s = k_s L_s \cos^e\beta = k_s L_s (\boldsymbol{v} \cdot \boldsymbol{r})^e$，其中 β 是反射光线 $\boldsymbol{r}$ 与人眼视线方向矢量 $\boldsymbol{v}$ 的夹角（如图5.9(b)所示），e 是高光系数。通常情况下，e 越大，表示物体表面越接近理想的镜面反射表面，镜面反射形成的高光越集中；e 越小，表示物体表面越接近漫反射表面，镜面反射形成的高光越分散。

如图5.10(d)所示，将上述三种光照效果进行综合，按照相应的系数做线性叠加，最终就可以得到Phong着色模型所产生的绘制效果。该模型可以写成如下表达式：

$$I = k_a I_a + k_d I_d (\boldsymbol{l} \cdot \boldsymbol{n}) + k_s I_s (\boldsymbol{v} \cdot \boldsymbol{r})^e \tag{5.3}$$

其中，k_a、k_d 和 k_s 分别是环境光反射系数、漫反射系数和镜面反射系数。

相比于Gouraud着色模型，Phong模型的特点是能够在多边形的边界处产生更为连续的视觉效果，适合光滑物体表面的绘制，尤其是具有显著高光区域的模型。但是，Phong模型的计算量明显比Gouraud模型大。这主要是由于除了向量点积运算，还涉及指数运算。

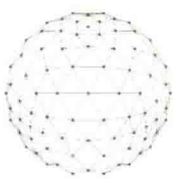


(a) 环境光照　(b) 漫反射光照　(c) 镜面反射光照　(d) 绘制效果

图5.10 Phong绘制效果

例题5-2 Phong着色模型。

问题：如右图所示的三角网格圆球面，写出采用Phong着色模型进行绘制的伪代码。



解答：根据指定的光源 L、材质 M 和视线 D，对每个△ABC 覆盖像素片元 f 进行着色的伪代码是

```
function Phong (L, M, V, f)
    for each fragment f on ball do
        (A, B, C) ← getTriangleVertice (f)
        normal ← normalTrilinearInterpolation (f, A, B, C)
        L ← getLight ()
        M ← getMaterial (f)
        D ← normalize (viewPos - f.pos)
        ambient ← L.ambient × M.ambient
```

```
        diffuse ← max (dot(-L.direction, normal), 0) × L.diffuse × M.diffuse
        specular ← pow (max(dot(D, reflect(L.direction, normal)), 0), M.shininess)
                × L.specular× M.specular;
        color (f) ← ambient +diffuse +specular
    end for
end function
```

□

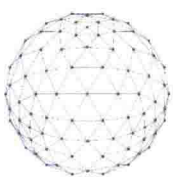
3. Blinn-Phong 着色模型

前面介绍的 Phong 着色模型实际上是一个基于几何的光照模型。在给定光照条件下，能够相对准确地模拟局部光照情况，产生近似于真实世界中光照作用下的绘制效果。但是，在计算公式(5.3)所示的 Phong 模型中的镜面反射效果时，需要对物体表面上的每点计算反射矢量 $\boldsymbol{r}$。而在早期计算机运算能力较差的情况下，逐点计算这种反射光线的矢量是相对比较耗时的任务。为了进一步简化计算，Utah 大学的 Jim Blinn(1999 年 Coons 奖获得者)建议采用近似方法计算镜面反射效果。这种近似的光照模型称为 Blinn-Phong 着色模型。该模型通过加/减法运算来计算中值矢量，在此基础上给出了近似镜面反射方向的计算方法，从而使得计算更加高效，同时绘制结果也能保持较好的真实感效果。

所谓中值矢量，是指位于观察者的矢量方向与光源的矢量方向之间的一个单位矢量 $\boldsymbol{h}$，表示为 $\boldsymbol{h}=(\boldsymbol{l}+\boldsymbol{v})/|\boldsymbol{l}+\boldsymbol{v}|$，如图 5.9(c)所示。那么，镜面反射中矢量点积$(\boldsymbol{v}\cdot\boldsymbol{r})^e$ 可以转化为$(\boldsymbol{n}\cdot\boldsymbol{h})^e$，这就避免了不断地反复计算反射矢量的过程。Blinn-Phong 模型能够在物体表面生成类似 Phong 模型的光滑着色效果。例如，当反射光线方向和人眼视线方向一致时，计算得到的 $\boldsymbol{h}$ 与 $\boldsymbol{n}$ 平行，镜面反射分量最大；当人眼视线方向偏离反射方向时，$\boldsymbol{h}$ 也偏离 $\boldsymbol{n}$，镜面反射分量变小。由于 Blinn-Phong 模型在计算时更加方便且快速，也成为 OpenGL 等很多图形流水线中普遍使用的绘制方法。

例题 5-3　Blinn-Phong 着色模型。

问题：如右图所示的三角网格圆球面，写出采用 Blinn-Phong 着色模型进行绘制的伪代码。



解答：根据指定的光源 L、材质 M 和视线 D，对每个$\triangle ABC$ 覆盖像素片元 f 进行着色的伪代码是

```
function Blinn-Phong (L, M, V, f)
    for each fragment f on ball do
        (A, B, C) ← getTriangleVertice (f)
        normal ← normalTrilinearInterpolation (f, A, B, C)
        L ← getLight ()
        M ← getMaterial (f)
        D ← normalize (viewPos -f.pos)
        H ← normalize (D +normal);
        ambient ← L.ambient × M.ambient
        diffuse ← max (dot(-L.direction, normal), 0) × L.diffuse × M.diffuse
```

```
        specular ← pow (max(dot(H, normal), 0), M.shininess) × L.specular×
M.specular
        color (f) ← ambient +diffuse +specular
    end for
end function
```

□

5.3 纹理映射

纹理映射(Texture Mapping),又称纹理贴图,是将纹理空间中的二维图像像素映射到三维模型表面,再转化为屏幕上二维像素的过程。与 Phong 着色等 BRDF 模型不同,纹理映射可以看作是把一幅图像直接贴到物体表面以增强其真实感,而不是直接去模拟光线和物体表面的作用效果。因此,纹理映射在绘制形状复杂的三维模型时具有更明显的优势,能够大大减少几何建模的工作量,降低整体的计算量,提高绘制效率。

通过纹理映射,能够在具有相同几何形状表示的情况下,达到更具真实感的绘制效果。同时,纹理映射结合着色技术,是一种很实用的真实感绘制技术。例如,绘制一个橙子的三维模型,如果是直接用一个橙色的球面显示,则显得过于简单,无法展现出真实橘子粗糙的表面。如果使用更复杂的几何形状,则需要模拟很多的几何细节,尤其是橙子表面的凹凸起伏。这就一方面大大增加了模型的复杂度,另一方面使得绘制时的计算量大为增加。

如果采用纹理映射的方式,就可以将一张相机拍摄的真实橙子表面的照片映射到简单球面的三维模型上。由于照片上橙子的真实性,在经过纹理映射后,就能够使该简单球面具有了真实橙子的外观。但是在有些情况下,这种直接的纹理映射效果仍然不是很好,因为其几何形状还是比较简单,而且呈现出的外观是不随视角而变化。进一步,如果需要体现橙子表面的几何细节的变化,可以采用纹理映射的另一种方式,也就是凹凸映射,能够实现更加真实的绘制效果。

常用的纹理映射方法主要有简单的纹理映射、环境映射和凹凸映射。简单的纹理映射,是使用图像像素填满多边形内部,从而将二维图像映射到由多变形表示的物体三维模型表面。环境映射,是使用环境照片来实现简单的纹理映射。凹凸映射,则是通过扰动表面法线矢量来改变表面的形状,从而达到丰富物体外观的目的。接下来具体介绍这三种纹理映射方法。

5.3.1 简单的纹理映射

简单的纹理映射是通过建立图像像素和物体表面多边形顶点之间的对应关系,从而把图像映射到物体表面。如图 5.11 所示,这种方法通过两个映射步骤实现:第一步映射是把二维图像坐标(也称为纹理坐标)变换到模型所在的三维空间的几何坐标;第二步映射是把物体变换到屏幕坐标系中。其中的关键步骤是第一步映射,在计算机图形学中称为参数化,也就是寻找映射函数将二维图像坐标变换为三维物体表面的坐标。一般情况

下，参数化映射是一对一的满射，因此也可以先建立三维模型到二维图像的映射，然后通过其逆映射来实现纹理映射。

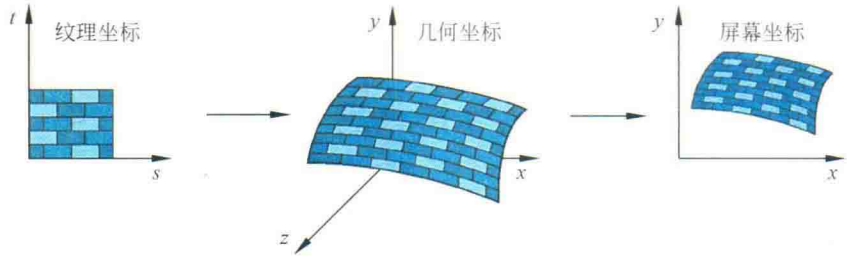


图 5.11 纹理映射的两步映射过程

线性映射是参数化最常用的方式，这需要首先建立图像到模型表面多边形的一一对应关系。例如，在图 5.11 中由左下角和右上角确定的纹理面片映射到多边形面片。那么，内部像素点所对应的多边形面片上的点就可以通过线性插值得到。数学上，这个过程是通过线性映射建立参数化函数来完成。这就需要对物体表面多边形的每个顶点指定相应的纹理坐标，记为$(x(s,t),y(s,t),z(s,t))$，其中(x,y,z)是顶点的三维坐标，(s,t)是对应的纹理坐标。

对于给定的任意形状的多面体模型，直接计算参数化函数比较困难，大多是基于数字几何处理中网格参数化的技术来实现的。我们在第 4 章中已经详细介绍了三维网格模型参数化技术。这里着重介绍另外一种较简单的参数化方法，称为两步映射，适合于一般图形流水线中的纹理映射。

两步映射的基本思想是根据物体三维模型的形状，首先选择一个较规则的中间形状，例如圆柱、球体、立方体等，将纹理图像映射到这些中间形状的表面。然后，将物体模型置于这些中间形状的内部，通过直接投影等方式建立物体表面和中间形状表面之间的对应。在此基础上，实现纹理图像到物体表面的参数化映射。

第一步是建立中间形状的纹理坐标。圆柱面是最简单的一种中间形状，其本质上是一种可展曲面。利用圆柱对物体三维模型的包围性，可以很方便地建立纹理图像像素和圆柱面上点之间的对应。假设纹理坐标的范围为 $s,t\in[0,1]$，而作为中间形状的圆柱面的高度为 h，半径为 r。那么，圆柱面上的点可以用如下参数化映射表示：

$$\begin{cases} x = r\cos 2\pi s \\ y = r\sin 2\pi t \\ z = t/h \end{cases} \tag{5.4}$$

在 h 和 r 固定的情况下，s 和 t 就可以作为纹理坐标。这样，如图 5.12(a)所示，就可以建立纹理图像和圆柱表面之间的一一对应。

球面也是一种简单的中间形状，很容易通过第 3 章中公式(3.2)所示的球坐标建立参数化映射函数(如图 5.12(b)所示)。此外，还可以采用立方体作为中间形状，把二维图像映射到一个打开的立方体表面。如图 5.12(c)所示，这个过程就类似于从一张纸板组装一个箱子。球面和立方体主要应用于环境映射。

(a) 圆柱面　　(b) 球面　　(c) 立方体面

图 5.12 第一步映射

第二步是要将中间形状的纹理坐标映射到物体三维模型表面。这个过程就需要建立中间形状和物体表面上点之间的一一映射。如图 5.13 所示，可以采用三种方式建立该映射。第一种方式选取中间形状上某一点的纹理坐标，沿该点的法向方向做反向延长，直到与物体相交。那么，该交点的纹理坐标就是中间形状上对应点的纹理坐标。第二种方式是上述的逆过程，选取物体表面上的一点，沿该点的法向方向延长，直到与中间形状相交为止。那么，该交点的纹理坐标就是物体表面相应点的纹理坐标。第三种是假设已知物体的中间位置，在该中心位置与物体表面上的某一点画一条直线，该直线与中间形状相交。那么，交点的纹理坐标就是物体表面上相应的纹理坐标。这三种从中间形状到物体表面的映射方式均适用于具有简单形状的三维模型，但是由于无法保证二者之间的双射关系，两步法不适合建立形状比较复杂的物体的纹理坐标。

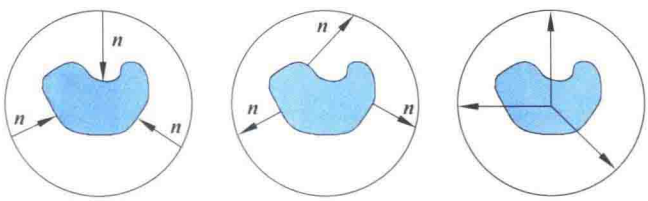
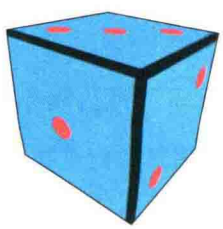


图 5.13 从中间形状到物体三维模型表面的映射

经过上述两步映射的步骤后，就可以建立物体表面和纹理图像之间的坐标变换，得到三维模型的纹理坐标，从而实现简单的纹理映射。

例题 5-4 简单的纹理映射贴图。

问题：采用 OpenGL 的纹理贴图功能实现如右图所示立方体表面的骰子效果。



解答：假设立方体的 8 个顶点组成的集合是 V，每个顶点对应的纹理坐标集合是 T，6 幅纹理图像存储于数组 texImg，那么纹理贴图的伪代码是

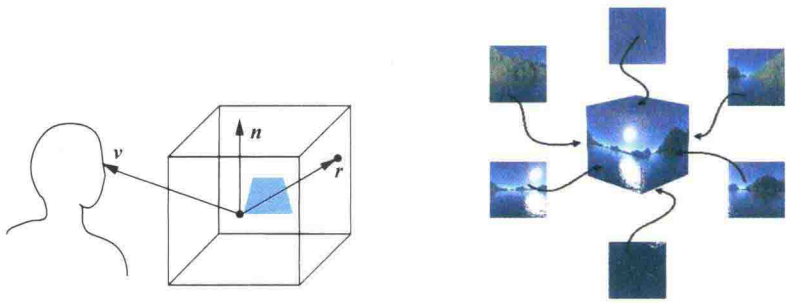
```
function texMapping (V, T, texImg)
    for i =0 : 5 do                          /*绘制每个面上的纹理图像*/
        P ← getVertexPos (V, i)              /*每个面的顶点序列*/
```

```
        C ← getVertexTextureCoord (T, i)    /* 每个顶点的纹理坐标 */
        T ← getCubeTexture (texImg, i)      /* 每个面的纹理图像 */
        glSetVertexArray (P, C)
        glBindTexutre (T)
        glDrawArrays (GL_TRIANGLES, 6)      /* 基于三角形的绘制 */
    end for
end function
```

□

5.3.2 环境映射

环境映射是利用物体所处的周围环境所形成的图像，构造物体模型的纹理映射。它通常用于模拟光滑表面对周围环境的反射。环境映射假设周围环境远离物体模型和观察者，同时物体模型不具备自身反射能力，而且空间中任意一点反射由其方向唯一确定。那么，就可以将周围环境作为纹理图像，通过纹理映射的方式投射到物体模型的表面。如图 5.14 所示，通过模拟人的眼睛对立方体内部的观察过程，将立方体表面的纹理图像映射到最终的绘制图像上，得到环境绘制效果。



(a) 眼睛对立方体内场景的观察　　(b) 场景图像映射到立方体表面

图 5.14　环境映射

环境映射方法主要包含三个基本步骤：①创建二维环境纹理图像；②将图像映射到立方体的上、下、左、右、前、后六个面或者一个完整的球面，以此作为纹理贴图；③计算物体模型表面的法向及反射方向，并通过反射方向采样立方体表面纹理贴图的颜色。总之，环境映射模拟了光滑物体表面(如镜子)对周围环境的反射，因而可以有效地绘制高度镜面的曲面。

5.3.3 凹凸映射

凹凸映射主要用来模拟物体表面几何形状的凹凸起伏。该方法的基本思想是通过扰动表面法向量，使得物体表面受光线照射的影响而产生视觉的变化，进而引起外观的改变。一般而言，凹凸映射解决的主要问题是如何使得物体表面看起来具有更自然的粗糙程度。对于该问题，直接的解决方法是使用很多小的多边形面片进行表面建模，但这样就会增加模型的几何复杂度，同时增加光照计算量。而凹凸映射则是仅通过扰动表面法向

量，在进行着色计算时模拟小的多边形产生的形状起伏效果，也就是生成视觉上假的凹凸效果。因此，凹凸映射最大的优点是不需要增加物体模型的几何复杂性，就可以很大程度地改变物体表面的外观。

如图 5.15 所示，凹凸映射主要通过两个步骤完成。第一步，对于输入的光滑表面计算法向量，并通过随机扰动改变法向，例如对法向向量的三个分量增加随机噪声。第二步，使用新的法向替代原始光滑表面的法向，进而计算新的光照效果。通常在第二步中采用局部着色技术，如 Phong 着色模型等，对扰动法向后的物体表面进行绘制，就可以得到具有明显凹凸效果的物体外观。这里需要注意，凹凸映射不会改变物体的几何形状，而只是“以假乱真”地使其绘制出的图像具有起伏不平的视觉效果。

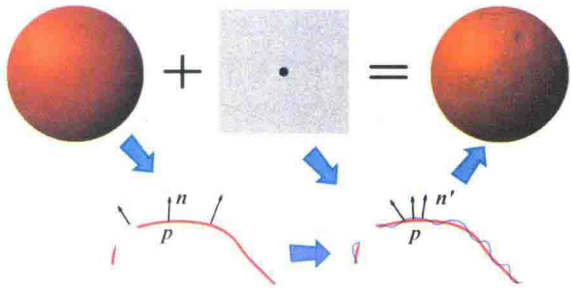


图 5.15 扰动物体表面法向生成凹凸映射效果

例题 5-5 凹凸映射。

问题：采用 OpenGL 实现右图所示的凹凸映射效果。

解答：假设组成球面的顶点集合是 V，三角形面片集合是 F，那么凹凸映射的伪代码是

```
function bumpMapping (V, F)
    for each fragment f in F do
        (A, B, C) ← getTriangleVertice (V, f)
        normal ← normalTrilinearInterpolation (f, A, B, C)
        uAxis ← calUAxis (normal, A, B, C)
        vAxis ← cross (uAxis, normal)
        T ← getTexture (f)
        gradient ← calGradient (T)
        normal ← normalize (normal +gradient.x × uAxis +gradient.y × vAxis)
        L ← getLight ()
        M ← getMaterial (f)
        D ← normalize (viewPos -f.pos)
        H ← normalize (D +normal)
        ambient ← L.ambient × M.ambient
        diffuse ← max (dot(-L.direction, normal), 0) × L.diffuse × M.diffuse
        specular ← pow (max(dot(H, normal), 0), M.shininess) × L.specular×
M.specular
```

```
            color (f) ← ambient +diffuse +specular
        end for
    end function
```

□

5.4 光线跟踪绘制

与 Phong 着色模型等局部绘制技术不同，光线跟踪(Ray tracing)是一种考虑光线和场景中所有物体表面整体作用效果的绘制模型。因此，光线跟踪是一种全局绘制技术。光线跟踪模型能够更加准确地模拟光线在场景中的传播，很容易绘制具有反射、折射、透射以及各种阴影效果的真实感图像，进而呈现高质量的真实感绘制效果。光线跟踪也被认为是计算机图形学历史上最为成功的真实感绘制算法之一。

纵观计算机图形学的发展历史，光线跟踪主要有两种方式：正向跟踪和反向跟踪。正向跟踪是模拟从光源发出的光线，经过与场景中不同物体表面的反射、折射、透射等作用，最终射入人眼的过程。反向跟踪则是从人的眼睛视点位置出发，只跟踪视野范围内能够看得到的物体表面对光线的各种作用。目前，几乎所有 3D 制作软件中的光线跟踪算法都采用反向跟踪方式。其原因是这种方式能够最大程度地节省计算机的存储和计算资源，同时不会降低绘制图像的真实感效果。接下来介绍基于反向跟踪的光线跟踪绘制算法。

5.4.1 基本原理

用于计算机图形学真实感绘制的光线跟踪算法最初是由 IBM 研究院的 Arthur Appel 于 1968 年提出的，当时被称为光线投射(Ray casting)。在这个模型中，Appel 只是将光线从眼睛位置往场景投射一束光线，然后根据交点处的光强进行绘制，而不再继续跟踪与物体作用后的光线。1979 年，Bell 实验室的 Turner Whitted 在其论文 *Recursive Ray Tracing Algorithm* 中提出，通过跟踪光线在场景中物体表面之间的多次反射以表现全局光照效果，从而获得更具真实感效果的图像。1984 年，美国 Lucas 影业公司的 Loren Carpenter 等人提出了分布式光线跟踪算法，提高了计算效率，有力地推动了真实感绘制技术的实用化。

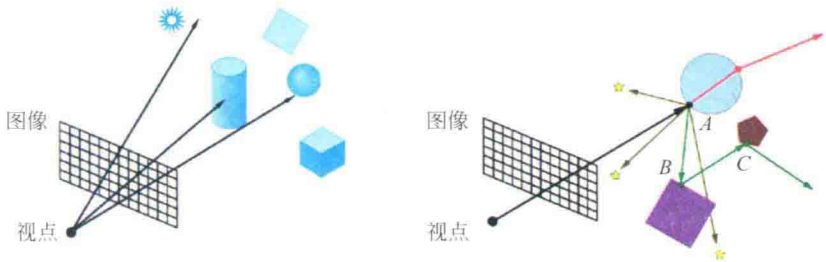
通俗地讲，光线跟踪的基本原理是物理学中的光路可逆。它从视点位置出发，向成像平面上的每一点(像素)发射光线。初始光线起始于人眼视点，终止于光线第一次与物体表面的交点。首先，对于成像平面上的每个像素点，可以通过跟踪光线找出与物体表面相交的点。然后，根据物体表面的材质属性，按照反射、折射等作用结果继续跟踪接下来传播的光线，并确定影响该点光强的所有光源。以此类推，就可以通过递归的方式计算光线与物体表面每个交点处的光强。因为初始光线对应于每一个像素，所以将该像素对应的所有跟踪光线的作用效果进行累积，就可以得到逐像素绘制的图像。

5.4.2 光线投射模型

事实上，光线投射模型是一种基于图像像素栅格化的模型。对于每一个像素，该模型从眼睛位置处投射一条光线，指向该像素点。在绘制场景图像时，跟踪每一条射线与场景中物体表面的作用。那么，每条射线一般存在两种情况：与一个物体表面或一个光源相交；或者不接触任何物体直接投射到无穷远处。因此，通过计算两种情况下的光强分布，就可以生成绘制后的场景图像。

如图 5.16(a)所示，进入眼睛的所有光线，有些直接来自于光源，那么就会受光源强度的影响；有些来自于物体表面的反射光线，那么就会受物体材质的影响。如果这些光线只经历了至多一次和物体表面的作用，则称为直接光。如果光线投射到无穷远，那么像素可直接赋值为指定的颜色，也就是绘制图像时的背景。当投影光线与物体表面相交时，就需要计算交点处的光强。例如，采用 5.2.2 节中的 Phong 着色模型计算交点处的光强。

综上所述，光线投射模型对于每一个像素点只需要计算一次直接光照射后的光强。因此，该模型计算效率高。但是，光线投射模型绘制的真实感效果不是很理想，无法准确地绘制镜面反射、阴影等需要光线和物体表面多次作用的光照效果。



(a) 光线投射　　(b) 光线跟踪

图 5.16　光线投射与光线跟踪

5.4.3 光线跟踪模型

光线投射模型和光线跟踪模型在早期并无不同。然而随着绘制技术的发展，更加有效的光线跟踪方法被提出。简单地讲，光线投射只对眼睛位置发射的光线做一次跟踪，并不会继续递归地跟踪光线；而光线跟踪则会继续跟踪物体之间反射或折射光线(如图 5.16(b)所示)。这些继续跟踪的光线就形成了间接光。

最基本的光线跟踪算法是跟踪镜面反射和折射后的光线传播，也称为光线跟踪器。一般的光线跟踪算法主要通过以下三个步骤的递归过程来完成。

(1) 对于待绘制图像的每个像素，沿视点位置和像素连线方向发射初始跟踪线到第一个可见物体表面，如图 5.16(b)中的 A 点所示。

(2) 根据物体表面的属性，按照反射、折射等方式生成第二条跟踪线，并以此递归，如图 5.16(b)中的 B、C 点所示。

(3) 根据递归结果，对相应的像素进行着色。

上述递归过程的伪代码如图 5.17 所示。其中，traceRay 函数实现了在反射和折射等情况下对光线和物体作用效果的递归调用。最终，通过对跟踪过程中每一点处光强的叠加得到像素最终的颜色值，完成该视点下对场景中所有模型的绘制。

```
function traceImage (scene):
    for each pixel (i,j) in image do
        S ← pixelToWorld (i,j)
        P ← COP
        d ← (S-P)/||S-P||
        I[i][j] ← traceRay (scene, P, d)
    end for
end function
```

```
function traceRay (scene, P, d):
    (t, N, mtrl) ← scene. inersect (P, d)
    Q ← ray (P, d) evaluated at t
    I ← shade (q, N, mtrl, scene)
    R ← reflectDirection (N, -d)
    I ← I + mtrl.k_r* traceRay (scene, Q, R)
    if ray is entering object then
        n_i ← index_of_air
        n_t ← mtrl.index
    else
        n_i ← mtrl.index
        n_t ← index_of_air
    if (mtrl.k_t>0 and notTIR (n_i, n_t, N, -d)) then
        T ← refractDirection (n_i, n_t, N, -d)
        VI ← I + mtrl.k_t* traceRay (scene, Q, T)
    end if
    return I
end function
```

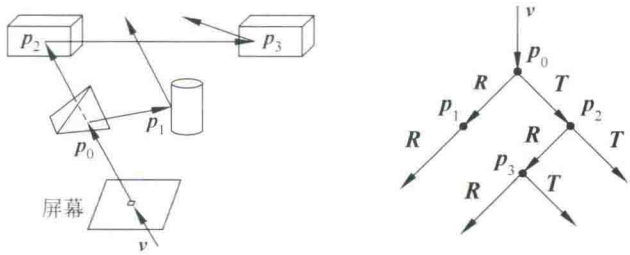
图 5.17 光线跟踪的伪代码

光线跟踪模型是对 Phong 着色模型等局部绘制方式的全局推广，也就是沿着跟踪的光线依次按照局部着色模型计算光线到达物体表面处的光强。光线跟踪的优点是能够更真实地模拟光线在空间中的传播过程，从而更自然地生成反射、阴影、景深、运动模糊等效果。光线跟踪算法思想简单、容易实现，同时绘制效果具有很强的真实感。其缺点则是算法复杂度相对较高，主要体现在需要递归地计算跟踪光线与物体表面的交点。

具体地讲，每一次光强计算都需要进行大量的光线与物体表面求交运算，而且需要同时跟踪大量的光线，由此形成一系列交点的序列。在光线跟踪过程中，光线沿着到达视点的光线的相反方向开始跟踪。每次经过屏幕上的一个像素，就计算跟踪线 v 与场景中物体的交点。这里的跟踪线既可以是从眼睛出发的视线，也可以是物体表面之间反射或折射后的光线。如图 5.18(a)所示，以距离最小的交点作为第一个可见交点 p_0。假设跟踪线 v 在 p_0 处产生反射或折射，那么就以所产生的反射光线或折射光线作为新的跟踪线，并且与物体表面求交得到新的交点 p_1。同时，产生新的反射线或折射线作为跟踪线。

上述递归不断地进行，直到所产生的跟踪线超出场景范围。最终，得到沿跟踪线所形成的轨迹上的一系列交点：p_0，p_1，…，p_n。如图 5.18(b)所示，这个过程可以表示为一棵

光线跟踪树，其中树的节点代表物体表面与跟踪线的交点，节点连线代表相应的跟踪线。每个节点的左儿子代表反射产生的跟踪线(R)，右儿子代表折射产生的跟踪线(T)。空箭头表示跟踪线射出场景。那么，光线跟踪的整个过程就可以通过这颗二叉树来描述，节点就是沿着跟踪光线的交点，而所有的叶节点就是那些最终射向光源或无穷远处的光线离开物体表面时的交点。



(a) 一条投射线的跟踪过程　　(b) 二叉树

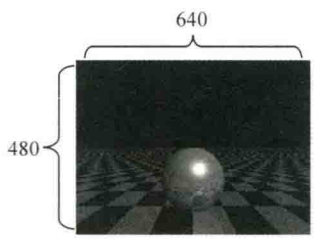
图 5.18　光线跟踪二叉树

物体表面 p_0 处的光强是 p_0，p_1，…，p_n 点光强的叠加。借助光线跟踪树，就可以采用后序遍历等方式快速访问树的节点，然后递归地调用局部绘制模型，进而计算出跟踪线方向的光强。此外，进一步按物体表面交点之间的距离进行衰减处理，然后再传递给父结点。由此上溯，得出 p_0 处的光强，也就是对应屏幕像素处的光强。

根据美国 Intel 公司测试报告，如果采用上述光线跟踪过程绘制出现代游戏的画面质量，且同时跑出流畅运行的帧数，每秒需要计算大概 10 亿束跟踪线。这个数字是由每帧每像素大概 30 束不同的光线所产生。在 1024×768 这样的入门级分辨率条件下，要实现每像素 30 束光线以及每秒 60 帧的要求，就需要每秒能运算 141.5 亿束光线的计算能力。因此，如何有效地提高光线跟踪效率，成为后期对光线跟踪算法进行改进的主要方向。

例题 5-6　简单场景的光线跟踪绘制。

问题： 如右图所示的简单场景，包含镜面光滑的球体和棋盘格平面地板。假设绘制图像的分辨率为 640×480，写出光线跟踪绘制的伪代码。



解答： 假设跟踪光线 ray 在第 n 次与场景中发生相交的物体是 intersectEntity，交点记为 intersectPoint，物体材质记为 M，而最大跟踪次数是 MAX_RECURSION，最后生成的颜色记为 I，那么伪代码如下。

```
function traceRay (ray, n)
    if tr >=MAX_RECURSION then
    end if
    (intersectPoint, intersectEntity) ← getIntersection (ray)
    if intersectEntity ==NULL then
    end if
    normal ← intersectEntity.calNormal (intersectPoint)
```

```
    M ← intersectEntity.calMaterial (intersectPoint)
    I ← M.kShade × shade (intersectEntity, intersectPoint, ray)
    if M.kReflect >0 then
        reflectDir ← reflect (ray.direcion, normal)
        I ← I +M.kReflect × traceRay (Ray(intersectPoint, reflectDir), tr +1)
    end if
    if M.kRefract >0 then
        if intersectEntity.rayEnterEntity (ray) then
            refractDir ← refract (ray.direction, -normal, M.index, airIndex)
        else
            refractDir ← refract (ray.direction, normal, airIndex, M.index)
        end if
        I ← I +M.kRefract * traceRay (Ray(intersectPoint, refractDir), tr +1)
    end if
end function
```

□

5.4.4 光线跟踪加速

通过上述分析可以看出，光线跟踪算法执行过程中的大部分时间是消耗在计算视线或光线与场景中物体表面的交点计算上，也就是确定光线跟踪二叉树的节点。因此，对光线跟踪进行加速，很大部分是针对求交运算的加速。根据场景中的物体几何形状，在进行光线与物体表面求交时，通常可以归结为如图 5.19 所示的三种情形下的求交运算。

1）情形 1. 直线与平面求交

这可以通过平面法向和直线方向建立显式表达式进行求解，能够非常快速地得到交点位置(如图 5.19(a)所示)。显而易见，一条直线和平面至多只存在一个交点。

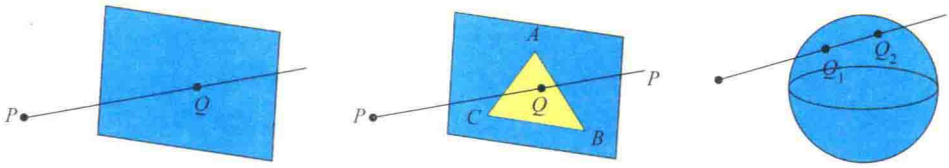
2）情形 2. 直线与三角形求交

这可以将其转化为直线与三角形所在平面的求交运算，得到交点后判断其是否位于三角形内部。如图 5.19(b)所示，如果位于三角形内部，则直线与三角形相交；否则，不发生相交。这种直线与三角形求交运算是转化为直线与平面求交以及三角形内部判断两部分，因而也是非常快速的。显而易见，一条直线和三角形也至多存在一个交点。

3）情形 3. 直线与球面求交

这可以将直线的参数表达式代入球面方程，转化为关于直线参变量的二次方程进行求解。然后，通过其判别式的正负决定是否发生相交。如果发生相交，较小参数对应的点是直线与球面的交点(如图 5.19(c)所示)。由于涉及二次方程求解，计算比前两种情形要复杂。显然，一条直线与球面至多有两个交点，但是在光线反射情况下，只有其中的一个交点用于后续光线跟踪。

大多数情况下，场景中的物体可以表示为平面、三角形或球面等基本形状组成的模型。那么，在进行加速时，常用的手段是减少光线与上述简单几何元素表面的求交次数。因此，可以通过对场景中物体表面进行合理的组织，避免不必要的光线与物体表面的求交运算，从而提高光线跟踪算法的效率。常见的物体求交加速技术包括层次包围技术、场景

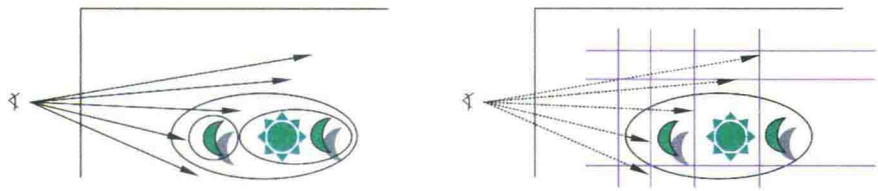


(a) 直线与平面求交　　(b) 直线与三角形求交　　(c) 直线与球面求交

图 5.19　直线与简单几何元素的求交

空间剖分技术等。前者利用物体包围盒对场景空间做均匀的划分，而后者则是对三维空间进行自适应剖分。两者都是借助分而治之的思想提高求交计算的效率。

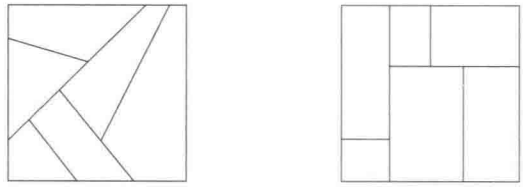
层次包围技术是建立场景中所有物体关于包围盒的树结构。包围盒是一种能够将物体包含在内的具有简单形状的几何体。例如，采用椭球或长方体将场景中的物体隔离。不同大小的包围盒形成树结构，其节点代表包围盒。在这种情况下，如图 5.20(a)所示，只有那些与父节点相交的跟踪线才会进一步与其子节点求交。通过这种方式，就可以预先排除不需要进行求交计算的场景空间，从而有效减少跟踪线与包围盒及物体求交次数。



(a) 层次包围技术　　(b) 场景空间剖分技术

图 5.20　求交加速技术

场景空间剖分技术是将场景中的物体按照它们在三维空间的位置分布，建立起层次结构组织，从而在光线与物体求交过程中能够快速剔除层次树上的无用分支，以此提高跟踪线与物体求交计算的效率(如图 5.21(b)所示)。这种自适应剖分技术本质上也是利用一种分而治之的思想，将复杂的求交运算做简化。如图 5.21(a)所示，常用的空间自适应剖分技术有 BSP 树、K-D 树等。BSP 树即二分空间划分树，是一类二叉树结构。它采用任意位置、方向的分割面递归地将空间划分为多个子空间对。BSP 树的根节点就是整个场景空间，而每个节点所代表的区域被平面分成两部分，一部分是平面前面(左侧)区域的子节点，另一部分是平面后面(右侧)区域的子节点。那么，场景中的物体就根据平面位置分属到各个节点。K-D 树算法是一种非均匀网格剖分算法，可以看作是改进的 BSP 算法。如图 5.21(b)所示，K-D 树的剖分平面是垂直于坐标轴的。在访问下一节点时，利用了跟踪线与节点边界相交的区域连贯性，以及一个辅助节点访问栈来进行遍历。通常只需要进行一次加法和一次乘法运算即可，因而具有较低的时间复杂度。通过 BSP 树或 K-D 树对场景空间进行剖分，就可以对跟踪线与物体相交情况进行预判，从而减少不必要的求交运算，有效提高光线跟踪的计算效率。



(a) BSP 剖分　　(b) K-D 树剖分

图 5.21　空间自适应剖分

发展到今天，大多数的影视级绘制系统都会集成基于光线追踪算法开发的真实感绘制技术。除了上述通过降低求交计算量的方式，很多基于硬件加速的光线跟踪加速技术也是不断涌现。例如 2005 年，德国 Saarland 大学的研究人员开发了第一个专门的光线跟踪硬件 RPU，以支持实时光线跟踪绘制。由于集成了专用的硬件单元，RPU 能够支持高速的递归函数调用，从而大大提高了光线跟踪的执行效率。此外，RPU 跟 GPU 一样都是完全可编程架构，提供了对材质、几何、光照等的实时编程支持。

5.4.5　光线跟踪的其他改进

传统的光线跟踪算法通过递归地追踪光线和物体的相互作用，能够生成真实度比较高的反射、折射和阴影效果。但是，由于在光线和物体作用时考虑的表面属性相对单一，且忽略了漫反射，使得绘制的光照效果往往比较僵硬，缺乏较细腻的柔化效果。针对这些问题，Caltech 大学的 Jim Kajiya 在 1986 年发明了蒙特卡洛光线跟踪(Monte Carlo ray tracing)算法，通过蒙特卡洛积分求解绘制方程，进而改进传统的光线跟踪绘制过程。其实，早在 Kajiya 正式提出蒙特卡洛光线跟踪算法之前，Cornell 大学的 Robert L. Cook 在 1984 年提出了具有类似思想的分布式光线跟踪(Distributed ray tracing)算法，也是通过对单个像素进行光线的重采样，实现具有软影、模糊等更加真实的绘制效果。这些都是通过更加细致地处理光线和物体表面的作用效果，进一步提高光线跟踪绘制的真实感程度。

简单地讲，蒙特卡洛光线跟踪算法就是通过概率论的方式计算光照积分的近似解，从而对光线和物体相互作用时的漫反射、镜面反射和折射分量分别进行建模。例如，当一条光线打到物体表面后，有 50%的概率发生漫反射、20%的概率发生镜面反射、30%的概率发生折射。这样，每次按照概率决定光线和物体的作用效果，就可以实现软硬、模糊等更多样的绘制效果。这样就避免了传统光线跟踪过程中要么物体表面没有被光线照射到，要么完全被光线照射到的情况，能够呈现更加细腻的柔化效果。换句话说，蒙特卡洛方法是将传统光线跟踪算法中每个像素追踪一条光线，扩展为每个像素追踪多条采样光线，最后通过积分求和来获得该像素的最终绘制效果。

5.5　辐射度绘制

虽然光线跟踪模型很好地模拟了光线在镜面或透明材质物体表面之间的传播过程，但它倾向于只在物体表面模拟一次光线反射或折射的效果，忽略了来自漫反射表面的反

射光线的影响。例如，两个位置相近、颜色不同的物体表面之间会产生“颜色辉映”现象。这种现象是由于表面反射光线的物体也看作能发射光线的光源所产生。因此，光线跟踪模型不能很好地模拟“颜色辉映”的光照效果。此外，光线跟踪绘制的阴影区域比较锐利，不符合日常光照效果。为了克服光线跟踪绘制算法的这些问题，1984 年，Cornell 大学的研究人员基于物理热力学中的辐射度模型，发明了辐射度绘制方法。这种方法通过模拟漫反射表面之间光线的多次相互作用，以实现场景的全局绘制。辐射度绘制能够生成更加贴合实际的柔和且自然的反射、阴影等效果。

5.5.1 基本原理

辐射度(Radiosity)源自于物理学中热辐射的概念，用于模拟单个或多个物体上两个表面之间的热量传输。辐射度在物理上描述物体表面向外发出的辐射通量密度，定义为单位时间内离开单位面积的能量，采用 W/m^2 作为计量单位。一般情况下，物体表面发射的能量主要是由两部分构成：物体作为发光体自身所散发的能量，以及反射外界入射的能量。其中，单位时间内从外界到达单位面积的能量称为辐照度(Irradiance)。考虑一个封闭的场景，使用具有单位面积的面片(Patch)来定义物体和光源表面。虽然每个物体表面的材质可能有所不同，但是它们都遵从相同的物理定律。简单地讲，物体表面对光的吸收、反射或折射都可以看作光线携带能量的变化，而光线经过表面之间的多次反射、折射等，最终达到一个能量平衡状态。那么，物体表面的光强实际上就反映了在平衡状态下场景中各处能量的分布。

如果把物体也看作是光源，那么离开表面的部分光是它自身所发射的光，其余部分的光则是入射到该表面的光的反射光。从能量守恒原理出发，物体表面辐射出的能量等于自身发射能量和反射能量之和。以空间中的 $\boldsymbol{p}$ 点为例，有如下能量表达式：

$$L(\boldsymbol{p}\rightarrow\vec{\omega}) = L_e(\boldsymbol{p}\rightarrow\vec{\omega}) + L_r(\boldsymbol{p}\rightarrow\vec{\omega}) \tag{5.5}$$

这里，$L(\boldsymbol{p}\rightarrow\vec{\omega})$ 表示沿方向 $\vec{\omega}$ 离开 $\boldsymbol{p}$ 点的能量，$L_e(\boldsymbol{p}\rightarrow\vec{\omega})$ 表示沿方向 $\vec{\omega}$ 自身发射的能量，$L_r(\boldsymbol{p}\rightarrow\vec{\omega})$ 则是沿方向 $\vec{\omega}$ 对吸收的入射能量的反射分量。这三项就反映了能量在物体表面之间传输时的变化及应该满足的平衡状态。

5.5.2 辐射度绘制模型

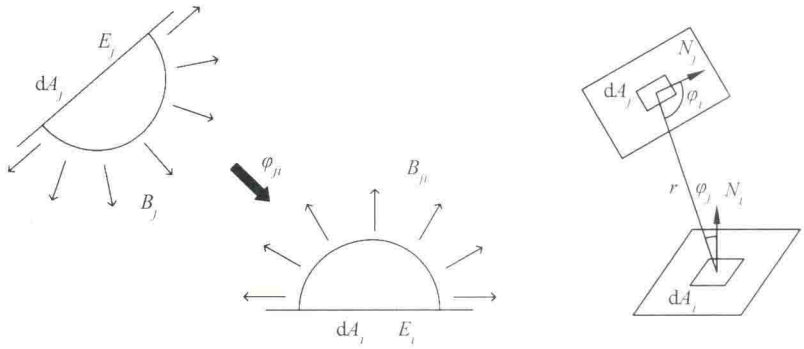
从辐射度的原理出发，场景绘制的结果就是由平衡状态下空间中光能量分布决定。假设空间中两点 $\boldsymbol{p}$ 和 $\boldsymbol{p}'$ 所对应的面片分别记为 dA_i 和 dA_j，而公式(5.3)中的方向 $\vec{\omega}=\overrightarrow{\boldsymbol{pp}'}$。那么，在 $\boldsymbol{p}$ 点的辐射度 B_i 应当满足如下等式：

$$B_i dA_i = E_i dA_i + R_i \int B_j F_{ji} dA_j \tag{5.6}$$

其中，如图 5.22(a)所示，E_i 是在 $\boldsymbol{p}$ 处面片 dA_i 由物体本身发射光线的辐射度，B_j 是在 $\boldsymbol{p}'$ 处面片 dA_j 向外传播的辐射度。F_{ji} 称为形式因子(Form Factor)，反映了从面片 dA_j 传播到面片 dA_i 的能量比重，也就是两个面片相互之间的能量影响，一般可以写为如下表达式：

$$F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j} \frac{\cos\varphi_i \cos\varphi_j}{\pi r^2} dA_j dA_i \tag{5.7}$$

这里，φ_i 和 φ_j 是两个面片之间光线与各自法向的夹角（如图 5.22(b)所示）。此外，公式(5.6)中的 R_i 是 $\boldsymbol{p}$ 点处材质因子，反映了物体对入射光线能量的反射率。公式(5.6)又称为辐射度方程，描述了面片上的能量分布情况。事实上，$B_i \mathrm{d}A_i$ 描述了单位时间内离开面片 $\mathrm{d}A_i$ 的能量，称为辐射通量(Flux)。



(a) 辐射度模型　　(b) 形式因子

图 5.22　辐射度模型和形式因子

根据辐射度方程，就可以计算出空间物体表面上每一点处的光强，以此生成场景的绘制图像。与光线跟踪等绘制方法相比，辐射度绘制的优点是能够非常真实地模拟漫反射表面光照，可以生成更加柔和、自然的阴影和反射效果，而且在计算时也容易使用图形硬件加速。但也存在一些缺点，例如不能很好地处理点光源和有光泽的物体表面，而且绘制模型相对复杂、整体计算效率较低等。因此，如何有效地求解辐射度方程，是辐射度绘制算法的关键。

5.5.3　辐射度方程的数值计算

数学上，辐射度方程是一个连续的积分方程。在实际绘制过程中，通常采用有限元方法进行计算。该方法将场景划分为许多面片，最终的辐射度是所有面片相互影响结果的叠加。因此，基本的辐射度绘制算法主要包含四个步骤：

(1) 将输入场景模型离散化为面片集合 $\{A_i\}$；

(2) 计算不同面片之间的形式因子 F_{ij}；

(3) 计算不同光线产生的辐射度 B_i；

(4) 将辐射度转换为颜色、亮度等属性进行场景绘制。其中，辐射度绘制算法时间主要消耗在第(2)步和第(3)步，分别占 90%的时间和近 10%的时间。因此，这两个步骤是影响绘制效率的重要因素。

1. 场景离散化

面片划分的粒度越细，对于辐射度方程的逼近越精确。为了保持物体表面离散辐射度的连续性，场景物体表面上每一点的辐射度都会通过对面片顶点的双线性插值进行计算。在实际计算时，如图 5.23(a)所示，场景往往离散化为面片表示，主要有均匀离散化和非均匀离散化两种方式。均匀离散化通常采用相同面积的规则面片进行划分，然后选

用常值或线性插值方法计算表面辐射度。显然，线性插值具有更好的连续性。非均匀离散化采用不同面积的面片，能够自适应地处理复杂场景模型。

2. 形式因子计算

这需要计算从面片 A_i 到达面片 A_j 的能量，其解析表达式较为复杂。这种情况下，通常采用半立方体方法近似计算。如图 5.23(b)所示，半立方体方法用半个立方体表面把两个面片分开，也就是将其中一个面片置于半立方体内部。半立方体表面被划分成规则的小正方形，这样面片很容易投影到这些小正方形上，进而得到每个面片所能接收到光能量的范围。然后，就可以利用投影区域计算相应的形式因子。假定要计算从 A_j 发出并到达 A_i 上 $\boldsymbol{p}_i$ 点的光能量，可以将 $\boldsymbol{p}_i$ 点作为中心放置一个半立方体，并划分成小正方形。在对由 $\boldsymbol{p}_i$ 点可见的小正方形区域计算能量值后，通过累加便可以获得面片 A_i 和面片 A_j 之间的形式因子。

(a) 场景离散化为面片　　(b) 半立方体形式因子计算

图 5.23　辐射度方程的数值求解

3. 辐射度计算

通过对面片光能量的离散求和获得每个面片的辐射度，可以写为

$$B_i = E_i + R_i \sum_j F_{ij} B_j \tag{5.8}$$

其中，B_i 和 B_j 表示彼此可见面片的辐射度。这样就得到了物体表面的光能量分布。

4. 场景绘制

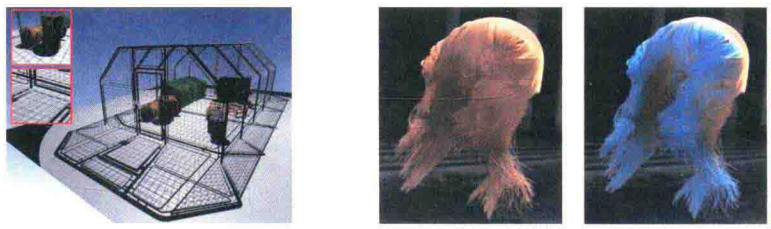
也就是将光能量转换为相应的颜色值、亮度值等。人的眼睛主要通过对 R、G、B 三个通道光能量的感知来区分不同的颜色。因此，辐射度绘制也就根据这三个通道光能量的计算结果，获得相应像素的颜色值、亮度值等。在此基础上，得到最终绘制的真实感图像。

5.6　特殊效果绘制

前面介绍的着色、光线跟踪、辐射度等模型，是针对一般几何模型和场景的绘制所提出的局部或全局技术，具有很强的通用性。它们可以实现大多数现实世界中的真实感绘制效果。而一些特殊效果的真实感程度，很大程度上取决于细节绘制的真实感。例如阴影的软硬程度、毛发的细微结构等。理论上，这些特殊效果也是可以在光线跟踪的框架下来绘制，但是由于效果本身的复杂性，往往需要耗费更多的计算资源，才能达到理想的真

实感效果，导致实用性不强。因此，针对这类特殊效果，往往需要设计更高效的绘制算法，才能更好地满足实际中真实感绘制的需求。接下来简单介绍针对几类特殊效果的绘制技术。

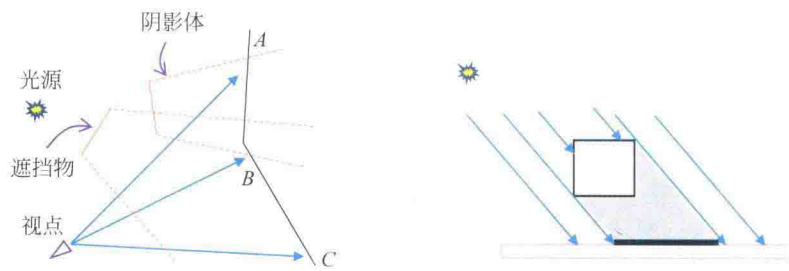
阴影是真实感绘制的重要内容之一，而绘制结果中是否包含阴影也直接影响了绘制的真实感程度。对于离线应用，光线跟踪或者辐射度方法都能够计算较准确的阴影，但其包含的求交等运算导致绘制效率低。为此，研究人员专门开发了阴影体(Shadow Volume)、阴影图(Shadow Map)等实时阴影绘制技术。它们都是借助专门的手段来减少对空间中的某一点是否位于阴影的判断，进而提高绘制效率，如图 5.24 所示。



(a) 阴影　　(b) 毛发

图 5.24　阴影和毛发的绘制(图片来自[42]和[43])

具体来讲，阴影体假设遮挡物为三角形，那么通过点光源和三角形的三个顶点连线就可以得到一个三棱锥(如图 5.25(a)所示)。进一步，将位于三角形平面光源一侧的顶部区域切除，就得到了阴影体。事实上，阴影体中的任何一点都是在阴影中。因此，要判断空间中的一个点是否处于阴影中，只需判断该点是否位于任意一个阴影体中。例如图 5.25(a)中的 A 点不在阴影区域，而 B 点和 C 点在阴影中。这样就能够大大减少传统阴影计算时的求交判断。阴影图技术则如图 5.25(b)所示，将视点放置于光源位置来观看场景，由此获得的绘制图像称为阴影图。那么，通过判断空间中的一个点是否能被光源“看见”，也就是是否位于阴影图中，就能判断该点是否处于阴影中。这样，通过对阴影图的预计算，就可以在绘制时提高阴影区域的绘制效率。



(a) 阴影体　　(b) 阴影图

图 5.25　基于阴影体和阴影图的绘制

毛发是一种分布稠密的物体，同时具有个体细小、呈半透明状等特性。这样，对毛发进行准确绘制，既需要模拟单根毛发的散射性质，又要考虑毛发之间遮挡产生的阴影。其中，单根毛发的散射效果通常采用毛发散射函数(Hair Scattering Function)描述，并将单

根毛发简化为单圆柱体模型进行计算。同时，采用双向散射近似法(Dual Scattering Approximation)来对毛发之间的多次散射进行快速模拟。毛发之间的遮挡会产生自阴影效果，但是由于单根毛发细小，且宽度小于一个像素，使得无法直接使用阴影体或阴影图技术绘制。这种情况下，一般是将视点放在光源位置观察毛发，同时记录下多层深度图；然后，毛发模型表面上每一点的阴影值通过从多层深度图中查询和插值得到。

5.7 小 结

真实感绘制是计算机图形学的重要内容。本章介绍了一些经典的真实感绘制技术。从光线和物体材质的作用，分析各种性质的光照模型以及 BRDF 模型。重点介绍了 Gouraud 着色、Phong 着色等局部绘制模型，光线跟踪、辐射度等全局绘制模型，以及从图像直接产生绘制效果的纹理映射技术等。此外，简单介绍了阴影、毛发等其他特殊效果的绘制技术。

虽然图形绘制的真实感效果在不断提升，但往往需要占用较多的计算资源，不利于手机等移动平台的真实感绘制。如何根据计算平台的性能特点，设计更加有效的真实感绘制算法，是当前计算机图形学在研究和应用方面面临的一个重要问题。

思考题

5.1 常见的光源模型有哪些？各自具有什么样的数学表示形式？

5.2 BRDF 是如何定义的？常用的 BRDF 模型有哪些？

5.3 Gouraud 着色模型和 Phong 着色模型的区别是什么？分别对绘制效果产生什么样的影响？

5.4 Blinn-Phong 模型是如何改进传统 Phong 着色技术的？

5.5 简单纹理映射的步骤有哪些？各自起到什么作用？

5.6 光线投射模型和光线跟踪模型的区别是什么？哪一个与真实的光照情况更吻合？

5.7 传统光线跟踪算法的改进方式有哪些？如何实现？

5.8 辐射度绘制的基本原理是什么？它解决了传统光线跟踪的哪些问题？

5.9 辐射度绘制方程中形式因子的作用是什么？

第6章 非真实感绘制

非真实感绘制(Non-Photorealistic Rendering, NPR)是指用艺术化的风格来表现真实世界中的场景,使场景内容能够按照一定的艺术风格进行展示。这是画家通过艺术作品反映客观世界的过程。那么对于计算机图形学中的非真实感绘制,其目的就是让计算机模拟艺术家绘画创作的过程,生成具有艺术效果的影像。因此,非真实感绘制是一种和真实感绘制具有相反意图的计算机图形学的绘制方式。

针对不同艺术风格图像的特点进行分析,本章着重介绍各种风格绘制模型,包括笔画建模、纹理合成、图像滤波等,以及基于这些模型的图像非真实感绘制技术。进一步,通过对这些绘制模型的时空优化处理,介绍视频的非真实感绘制技术。此外,结合当前深度学习方法的应用,介绍基于神经网络的非真实感绘制技术。

6.1 概　　述

非真实感绘制最早开始于20世纪80年代的计算机图形学研究。1986年,MIT的Steve Strassmann提出了一种使用具有一定宽度、但形状细小的毛刷模型进行图形绘制的方法,而不是以往真实感绘制时的光照和材质模型。这样绘制出的图像能够模拟西方艺术中的油画效果,由此揭开了非真实感绘制的研究进程。事实上,"非真实感绘制"这个词最早是由Washington大学的Georges Winkenbach和David Salesin在1994年SIGGRAPH的论文中首次使用的。他们在论文中提出了通过计算机程序来创作具有钢笔墨水风格图像的方法,并且能够将整个的创作过程做半自动的处理。

非真实感绘制本质上是要反映人类主观参与的艺术创作形式,这与反映物理规律的真实感绘制有着本质区别。从绘制过程来看,真实感绘制涉及复杂的几何和物理计算,往往需要设计特殊技巧的算法,以实现尽可能逼真的绘制效果,例如第5章介绍的光线跟踪、辐射度等绘制技术。非真实感绘制算法对复杂度的要求往往没有那么高,只需要能模拟使用特定艺术工具进行各种风格作品创作的过程,例如画笔、钢笔、墨水、颜料、布料、纸张等的使用。图6.1展示了对于同一个几何模型,分别使用真实感绘制技术(例如纹理映射)和非真实感绘制技术(例如素描)得到的绘制效果。这两种方式都是对三维模型外观的展示,但侧重的视觉效果和信息反馈却截然不同。简而言之,对于图6.1(a)中的三角网格模型,既可以采用图6.1(b)所示的纹理映射方式增加模型的真实感效果,也可以采用图6.1(c)所示的素描风格化方式使其展现艺术化的视觉效果。用户观看这两种效果时,直接能接收到的感观信息也会有所不同。

(a) 网格模型　　(b) 纹理映射　　(c) 素描风格化

图 6.1　同一个几何模型使用真实感绘制(纹理映射)和非真实感绘制(素描)的效果

从实际应用的角度来看，真实感绘制主要面向真实场景的记录，或者是对物理过程的客观模拟和仿真，例如水流、海浪、烟雾等自然现象的计算机展示。非真实感绘制则广泛应用于对特定事物的主观描述、个人情感宣泄的解释说明，例如在故事书中使用手绘插图描述情节或者机械元件的图解说明。因此，非真实感绘制通常是将模型进行抽象以传达重要的信息，以此突出真实影像中不容易觉察的认知信息，或者是增加展示的趣味性和艺术性，而非客观的真实性。

图 6.2 展示了非真实感绘制技术的发展历史。早期的非真实感绘制过程非常依赖于用户输入作为辅助，半自动地生成具有某种艺术风格的图像。1998 年，New York 大学的 Aaron Hertzmann 提出了完全自动的绘制方法，极大地推动了非真实感绘制的发展进程。此后，越来越多的图像处理和计算机视觉方法被引入到非真实感绘制。进入 21 世纪后，以机器学习为代表的人工智能技术的蓬勃发展，使得更高级的语义信息，如视觉注意机制、物体识别等可以被引入到非真实感绘制过程中，再加上提高绘制速度的 GPU 广泛使用，使得非真实感绘制发展更为迅速，在实际应用中也取得了长足的进步。非真实感绘制

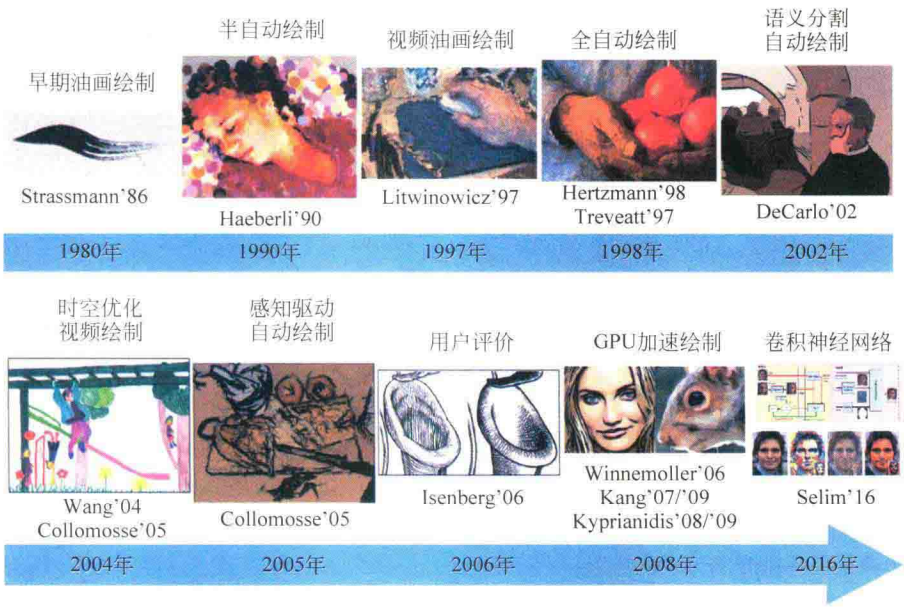


图 6.2　非真实感绘制技术的发展

除了生成艺术风格化的图像，也能够对视频帧画面进行处理，生成艺术风格化的视频。最近一些年，随着卷积神经网络和生成对抗网络等深度学习方法的出现和应用，非真实感绘制技术呈现出更加多样和实用的发展趋势。然而，非真实感绘制的核心，仍是力求模拟人类参与的艺术创作的过程，也就是根据艺术效果的风格特点建立相应的模型，以此启发不同绘制技术的研究。

6.2 基于笔画建模的绘制

笔画是艺术家使用画笔进行创作时所产生的表现形式，是构成油画、水彩画、素描等艺术作品的基本元素。因为笔画通常是由创作者使用各种类型的画笔所产生的，所以画笔及相应笔画的属性直接决定了绘画效果的风格差异。如图 6.3 所示，油画画笔通常具有扁平头形状，画出的笔画模样与圆头的水彩笔、纤细的钢笔和铅笔等是不同的。此外，即使是同一类型的画笔，在使用时施加不同的尺寸、方向、颜色、浓度等手法，也都会影响绘画效果。因此，如何建立合适的笔画模型是基于笔画进行非真实感绘制的重要环节。笔画模型既要能够反映该类艺术风格的主要特点，也要具有一定的灵活性，也就是能够根据场景内容的差异做相应的模型调整，并能够同时选择不同的笔画生成方式。这样才能体现艺术创作时的主观因素，使绘制效果更贴近真实的艺术创作。

(a) 不同形式的艺术创作工具　　(b) 非真实感绘制的笔画模型

图 6.3 模拟艺术创作的笔画模型

6.2.1 笔画模型

基于笔画建模的非真实感绘制方法，需要对各种类型的笔画分别进行建模。笔画模型应当既包含描述笔画绘制时相应的尺寸、形状、方向等几何方面的参数，又具有颜色、亮度、浓度、纹理等色彩方面的参数。这样在非真实感绘制过程中，根据输入图像内容本身的颜色、纹理等属性，指定笔画模型相应的参数，进而生成指定风格的绘制结果。此外，在建立笔画模型时，为了进一步提高绘制后的视觉效果，也允许加入一定量的用户交互，指导笔画模型创建时的正确性和绘制时的合理性。

简单来讲，一般的笔画可以用符号 $S(\boldsymbol{p},l,w,s,\boldsymbol{d},c,i,t)$ 来描述（如图 6.3(b)所示）。这也是笔画模型包含的主要参数，能够描述绝大部分艺术创作时画笔产生的笔画模样，例

如油画、水彩画、素描画等。具体来讲，参数 $\boldsymbol{p}$、l、w、s、$\boldsymbol{d}$、c、i、t 分别定义了笔画的中心位置、长度、宽度、形状、方向、颜色、浓度和纹理信息。具体的参数数值及其相应的笔画模型，需要根据输入图像和指定的艺术风格特点进行设置。这些笔画按照一定的顺序依次在画布上叠加，以模拟画家在绘画创作时逐笔绘制场景的过程，最终生成相应风格的绘制效果。

接下来，针对油画、水彩画、素描等不同画种的艺术形式，介绍相应的笔画模型以及各种风格化绘制方法。

6.2.2　油画风格化绘制

油画起源于欧洲，大约是在十五世纪时由荷兰人发明的。油画是西方绘画史中的主体绘画方式，在西方艺术史中占据着重要地位。传统油画采用亚麻子油调和颜料，在经过处理的布料或木板上作画。由于油画所使用的颜料不透明，覆盖力和黏着力强，所以油画绘制时可以产生由深到浅、逐层覆盖的笔画效果，从而使绘制的画面往往具有很强的立体感。

图 6.4 展示了两幅著名艺术家的油画作品：莫奈的《日出》和列宾的《伏尔加河上的纤夫》。从这两幅画中就可以明显看出现实中艺术家创作油画艺术作品时所运用笔画的一些特点。例如，由于油画画笔和颜料的特性，使得这些油画通常颜料黏稠，导致在画布上的笔画具有拖摆感和层次感；笔触可粗可细、可大可小，再结合颜色运用，就能够同时表现出多种风格，例如写实、印象、人物、风景等。因此，油画是一种典型的基于笔画绘制的艺术形式，在进行非真实感绘制时，主要解决的问题是各种笔画参数的设置，并且能够正确地模拟笔画覆盖过程，从而生成具有真实油画特点的绘制结果。

图 6.4　画家创作的油画艺术作品

代表性的绘制方法包括交互式笔画绘制、半自动笔画绘制、自动笔画绘制、笔画迁移绘制等。接下来分别介绍这些不同的绘制方法。

1. 交互式笔画绘制

早期的风格化绘制时，笔画往往需要借助人工交互的方式，来引导笔画参数设置以及这些笔画在画布上依次叠加的绘制过程。常见的交互方式是借助鼠标、键盘等输入设备的操作来定义笔画模型的相应参数。这样就可以通过预设的交互行为对输入的图像构建交互式的笔画模型。

如图 6.5 中所示，笔画的颜色 c 通过读取鼠标点击输入图像位置处的像素颜色来确定；笔画形状 s 采用简单线段来表示，而其方向 $\boldsymbol{d}$ 则由鼠标在屏幕上滑动的方向来定义，长度 l 则根据鼠标滑动距离来定义；笔画的宽度 w 通过键盘上下键来控制（例如上键加粗或下键减小宽度）；笔画的浓度 i 则由鼠标单击时间长短来定义。这种主动的鼠标和键盘交互方式，使得用户可以根据对输入图像内容的理解以及对油画风格的认知，主观地评判并设置相应的笔画模型。而在将这些笔画变成油画图像的具体绘制过程中，往往是以输入图像为模板，然后类似临摹的过程，在输入图像上面滑动鼠标和敲击键盘依次添加笔画，层层覆盖，最后将输入图像转化为具有油画风格的图像，实现交互式的笔画绘制。这里需要注意，在交互过程中并不需要指定每个像素点处的笔画，而是可以通过插值一些关键点处的笔画来提高交互式笔画绘制的效率。

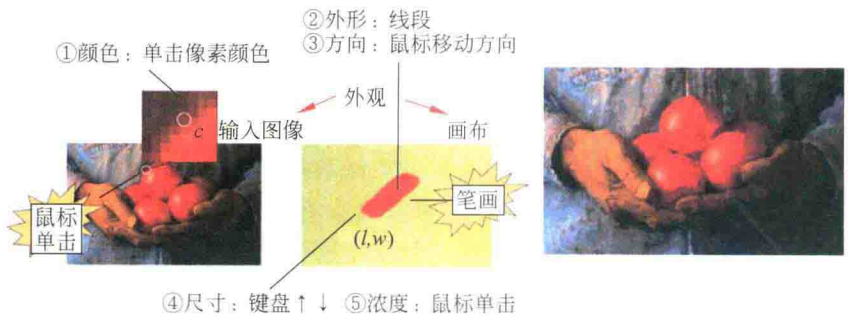


图 6.5 交互式笔画绘制中的模型参数设置和绘制效果

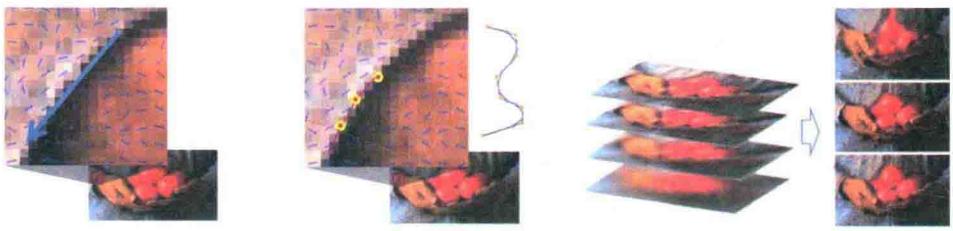
2. 半自动笔画绘制

在上述交互式笔画绘制时，有些笔画参数完全可以从输入图像自动进行计算，而不必都通过交互产生。这样能使绘制后的图像更准确地反映输入图像的特点，同时也能够提高笔画绘制效率。例如，笔画的长度 l 和宽度 w 可由图像每一点处的像素梯度来定义，记为 $l=\alpha|\nabla I|$，$w=\beta|\nabla I|$，这里 α 和 β 是预设的比例因子，用于控制实际绘制时笔画的形状。这是因为一般梯度较大的地方，往往是物体边缘特征的位置，所以可采用更突出的参数设置来绘制，有效地保持图像中的物体外观。此外，笔画的浓度 i 可以在鼠标单击时通过随机数来生成，其目的是为了增加人为创作的主观意图，从而能够更贴近于艺术创作的主观性。对于笔画模型的其余一些参数，则仍旧通过交互方式来指定。总体而言，半自动笔画绘制可以提高图像整体的绘制效率，但其中随机产生的笔画浓度有时就会影响图像中的局部细节，不利于艺术创作时美学特点的展现。

3. 自动笔画绘制

前面介绍的两种交互式和半自动笔画绘制方式还是需要手工交互设计参数来引导笔画的建模，而且在绘制过程中模型很难自适应地调整这些参数，从而导致风格化绘制时的灵活性比较差，艺术表现力不足。此外，这两种方式中的笔画形状多为简单的线段，也会导致绘制的效果比较生硬。针对这些问题，研究人员相继提出了自动笔画绘制方法。这类方法能够自动完成笔画模型参数的设置，而且采用更加复杂的笔画形状取代简单的线段，是一种更为实用的油画风格化绘制方法。

自动笔画绘制时往往采用图像处理中的边缘检测结果来定义笔画方向 $\boldsymbol{d}$，这样就可以更好地模拟画面中物体绘制时的笔画走向。具体来讲，首先通过均匀随机采样确定笔画的起点，然后通过该点处的 Sobel 算子响应获得物体的边缘分布（如图 6.6(a)所示），以此作为相应的笔画方向。对于笔画形状，则如图 6.6(b)所示，采用 B 样条自由曲线代替交互式或半自动绘制时使用的简单线段。这样在边缘检测得到局部特征方向后，就沿相应的方向通过间隔设置控制顶点进行拟合，生成由 B 样条曲线定义的笔画。显而易见，由于 B 样条自由曲线的光滑性和局部可控性，由其定义的笔画能够准确地描绘物体形状，从而提高油画绘制的写实性。



(a) Sobel 算子定义的笔画方向　(b) B 样条曲线定义的笔画形状　(c) 笔画逐层覆盖

图 6.6　自动笔画绘制的笔画模型和逐层覆盖过程（图片来自[47]）

例题 6-1　基于简单线段笔画模型的自动绘制。

问题：如右图所示的输入图像（上图）和风格化绘制结果（下图）。假设笔画模型的形状为线段、方向为图像梯度朝向、宽度为图像梯度的大小、颜色为均匀采样像素颜色，其他参数可默认取值，试写出自动笔画绘制的伪代码。

解答：输入图像 I 的宽度和高度分别是 w 和 h，在像素 pix(x, y)点处的梯度方向记为(grad_x, grad_y)，梯度大小记为 thick，采样像素颜色记为 color(x, y)，线段 line 默认长度为 6。那么采用该线段作为笔画模型的自动绘制伪代码如下所示。

```
function autoDraw (I, w, h)
    for pix (x, y) in I do
        for x ← 0 to w-1 do
            for y ← 0 to h-1 do
                grad_x ← pix (x+1, y)-pix (x, y)
                grad_y ← pix (x, y+1)-pix (x, y)
                thick (x, y) ← sqrt (grad_x^2 +grad_y^2)
                line_start ← pix (x, y)
                line_end_x ← x +6 * grad_x / thick (x, y)
                line_end_y ← y +6 * gradient_y / thick (x, y)
                line ← line (line_start, line_end, color (x, y), thick %4)
                y ← y +6
            end for
            x ← x +6;
```

```
        end for
    end for
end function
```

□

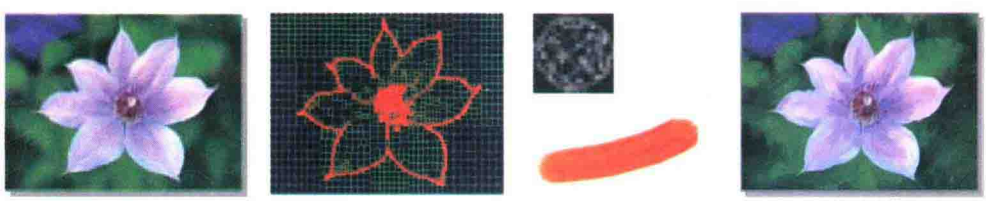
基于上述笔画模型，接下来就可以在绘制时采用分层的策略，由粗到细、由大到小地逐层叠加笔画，从而更好地体现油画创作手法中的由深到浅、逐层覆盖的艺术特点。具体来讲，利用高斯金字塔滤波对输入图像进行多尺度处理，形成由粗到细的分层。那么，在粗的分层选择尺寸较大的笔画，而在细的分层则选择尺寸较小笔画。将不同分层笔画绘制的结果按照层级进行顺序叠加，并根据浓度调整不同笔画颜色的混合效果，生成最后的绘制结果（如图 6.6(c)所示）。

上述自动绘制方法完全是从输入图像的底层视觉特征入手，采用图像处理的手段建立相应的笔画模型和绘制过程。这类方法虽然能够较好地生成具有油画特点的绘制结果，但缺乏足够的艺术表现力和生动性。这里的主要原因在于油画绘制本质上是一个人类主观创作的过程，创作的油画作品饱含画家的思想感情。因此，将图像高层语义特征加入到笔画建模和绘制过程中，将能够大大提高绘制的效果。基于这种思路，往往就需要利用输入图像中的语义信息。典型的方法有结合视觉显著性的自动绘制、结合语义分割的自动绘制等。

4. 结合视觉显著性的自动绘制

视觉显著性用于描述人类视觉注意力的特点，尤其是复杂场景下的人眼观察机制。因此，可以利用图像边缘显著性分布，指导笔画模型中的参数设置和绘制过程，这样就能够使得笔画建模和绘制过程更符合人的创作活动特点。

该方法首先计算输入图像的视觉显著性分布。最简单的方法是通过图像局部梯度的大小来表示显著性。那么，梯度分布对应的显著性分布图就决定了笔画覆盖时不同参数对结果的影响程度。如图 6.7(b)所示，这是一个非均匀的网格图，网格的大小表达了显著性强弱。该显著性分布是将整个画布划分成规则矩形组成的网格，然后沿 x、y 方向不断二分，直到生成的每个新格子的总显著性小于给定的阈值，从而得到最终的显著性分布图。该分布图实际上也反映了人眼注意力在图像上的集中情况。一般而言，越显著的区域对应的注意力就越集中，也就更需要精细的笔画进行绘制。因此，以该分布图作为指导，能够在绘制过程中随时调整笔画参数，绘制出符合显著性分布的结果。



(a) 输入图像　(b) 视觉显著性分布　(c) 笔画模板　(d) 绘制结果

图 6.7　结合视觉显著性的自动绘制（图片来自[48]）

对于笔画长度 l 和宽度 w，需要利用显著性的强弱，指定相应的两个参数。例如，对

于显著性强的格子使用长度短、宽度窄的笔画以体现细节特征。对于笔画的位置 $\boldsymbol{p}$，则根据格子显著性强度定义采样点个数，以此决定该格子内部笔画位置分布，使得在同一个格子内的笔画也能具有多样性。此外，对于笔画的形状 s，则改进了以往自动方法所使用的预先设计的反走样笔画，而是使用一个各向异性笔画模板（如图 6.7(c)所示）作为基本笔画形状。这样就能够一定程度地模拟画笔和画布之间的压力效果，也就是用笔越有力就会产生越明显的笔画痕迹。这种笔画在不同部位的颜色虚实不同，因而呈现出一定的纹理效果，同样也提高了笔画绘制时的多样性。

基于上述绘制方法，就可以更好地保留输入图像中的显著特征，得到的风格化结果更具有画家手工绘制的视觉效果。图 6.7(d)展示了结合视觉显著性的自动绘制结果，从中可以看出很明显的油画笔刷逐层覆盖的效果。此外，对于输入图像中视觉非显著的部分区域（例如绿叶部分）可以通过较粗的笔画绘制，而对于显著区域（例如花蕾部分）则要进一步使用较细的笔画进行绘制。这样的绘制结果也符合画家进行创作时的笔画运用规则。总体而言，这种绘制方法使用简单的视觉显著性就能够很明显地提高笔画绘制的效果，而且整个过程可以自动完成。

5. 结合语义分割的自动绘制

视觉显著性加强了注意力对油画风格化绘制效果整体感受的影响，使其能够更好地符合油画在创作过程中的主观意图。然而，艺术家在绘制油画时也需要根据场景中物体对象差异来施加不同的绘制手法，使其具有更丰富的多样性。因此，从语义角度对输入图像进行分区域的绘制，将能够进一步提高绘制效果。

语义，是通过认知获取数据的蕴含信息，也就是将图像作为视觉载体时潜藏的意义。在非真实感绘制时结合语义分割，其基本思想是利用不同尺度的图像视觉特征构建场景的结构化组织，从而挖掘符合人类主观感受的语义信息，进而指导笔画建模和绘制过程。语义解析树是从输入图像挖掘语义信息的基本工具。该解析树是对图像内容中包含的各个组成部分构建层次化的结构表示，以此来获得图像蕴含的语义成分（如图 6.8(a)所示）。

具体来讲，在绘制的过程中，输入图像首先通过一个分层的图像解析步骤，将其表示为一个从粗糙到细致的层次结构。这种层次结构就表示为图像的解析树。该解析树的节点表示图像中语义相近像素聚集而形成的不同的视觉模式，例如：①纹理区域，如天空、水、草、土地等；②表示轮廓线或线状结构的曲线，如树的树枝、栏杆等；③对象，如头发、皮肤、面部、衣服等。显而易见，这些节点具有很明显的语义信息，从而能够更好地辅助笔画建模。

基于上述解析树，笔画模型中的笔画方向 $\boldsymbol{d}$ 可以通过不同区域边界的特征线形成的方向场来定义。笔画形状 s 和浓度 i 等参数，则可以通过预先定义的笔画字典来选择。如图 6.8(b)所示，该字典包含了事先由艺术家通过手绘而画出的不同弯曲度、浓淡的笔画。笔画颜色 c 则基于对艺术家用色习惯进行统计而设置，通过直方图匹配使得绘制的图像尽可能满足艺术家用色的特点。在此基础上，通过语义解析树的层次结构对不同区域采用相应参数的笔画进行逐层覆盖，最终生成具有油画效果的风格化图像。总体而言，这种自动绘制方法充分利用图像构成方面的语义信息，同时结合大量艺术家手绘笔画作为素材，进一步提高了自动绘制的风格效果。

(a) 输入图像的语义解析树　　(b) 笔画字典

图 6.8　语义分割和笔画字典(图片来自[49])

6. 笔画迁移的自动绘制

无论是交互式笔画绘制,还是(半)自动笔画绘制,都是以输入图像作为参考,借助预设的笔画模型生成风格化的绘制结果。事实上,油画发展的历史中保留下来了大量的艺术家创作的油画作品。因此,可以通过这些真实的油画作为参考图像,直接将输入的图像转化为具有相同或相似风格的绘制结果,而不再是从一笔一笔的笔画叠加来生成绘制结果。这种思路就启发了基于风格迁移的非真实感绘制方法。因此对于基于笔画的油画绘制,另一种可采用的方式是笔画的迁移。

总体而言,笔画是反映油画风格的最主要特征。那么,就可以拿上述的真实油画图像作为模板,先从给定的模板中提取出笔画模型所涉及的相关参数,然后将这些参数迁移到输入图像的笔画模型进行绘制。显而易见,按照这种思路绘制出的图像就继承了笔画所来自的真实油画的风格。

给定模板油画图像,首先通过笔画检测技术寻找图像中笔画比较明显的区域,并建立笔画置信度图描述不同像素属于笔画覆盖区域的概率。这个过程也就是为从油画图像自动构建笔画模型提供候选区域。然后,对笔画区域进行 $S(\boldsymbol{p}, l, w, s, \boldsymbol{d}, c, i, t)$ 的参数分析,也就是借助图像处理的手段来反求相应的笔画参数。确定笔画模型中各种参数的数值是笔画迁移方法中最关键的步骤。接着,将这些参数转移到输入图像,确定绘制输入图像时所使用的笔画模型。最后,利用自动笔画绘制方法生成非真实感绘制结果。

具体来讲,对于模板油画图像,笔画痕迹大多显现在纹理区域,而且具有较强的局部方向性。因此,可借助 Garbor 滤波器等来实现笔画的自动检测。一般认为,Gabor 滤波器在处理图像时,对频率和方向的表达与人类视觉系统有很高的相似性。不同空间频率的 Garbor 能量响应能够被用来处理不同的笔画。例如频率越大,响应图像的显著轮廓线就越少,从而也能够对图像中不同显著性的物体进行区分。借助 Garbor 响应结果,就可以生成笔画置信图,用于表示该响应检测到的笔画分布(如图 6.9 所示)。这样通过不同频率下的响应,就获得了不同尺寸的笔画。

然后,对检测到的笔画进行参数分析并建立相应的模型。通过对笔画置信度图进行

图 6.9 模板油画的笔画置信图(图片来自[50])

二值化处理,统计聚类后的由像素连通性获取到的各个聚类的平均面积,以此求解笔画长度 l 和宽度 w。笔画方向 $\boldsymbol{d}$ 也在很大程度上影响着油画风格。这里通过笔画置信度图计算图像的方向场,以此作为对应位置上的笔画方向。笔画浓度 i 反映了模板油画图像中是否具有明显的笔画痕迹,可以通过计算笔画置信度图的局部平均响应强度来获得。其余笔画参数则来自于输入图像自身的一些特征,进而建立相应的笔画模型。接着,就可以将这些提取到的笔画参数迁移到输入图像,并根据前面介绍的自动笔画绘制方法对输入图像进行笔画建模和绘制。最终,绘制出的图像就可以通过迁移的笔画表现出相应的风格,达到风格迁移和图像风格化绘制的目的。

图 6.10 展示了基于笔画迁移的绘制结果。这里输入一幅风车图像,然后根据不同的模板油画图像,将各自风格的笔画模型转移至输入图像,最后利用笔画绘制得到该风车图像对应的非真实感绘制结果。从这个例子可以看出,这种基于笔画迁移的方法能够批量生成多种风格的图像。

(a) 模板油画图像

(b) 绘制结果

图 6.10 笔画迁移绘制结果(图片来自[50])

6.2.3 水彩画风格化绘制

水彩画是用水调和透明颜料进行作画的一种绘画方法，简称水彩。由于水彩颜料的色彩透明性，使得一层颜色覆盖在另一层上的时候可以产生特殊的光感效果。因此，水彩画风格效果大多也是基于笔画建模来绘制。但是，由于在绘画过程中纸张上水的流动性，造成了水彩画不同于油画等其他画种的外表风貌和创作技法（如图 6.11 所示）。因此，在对这种水彩风格进行非真实感绘制时，还需要考虑液体、纸张等因素对绘制效果的影响，以建立更准确的笔画模型。

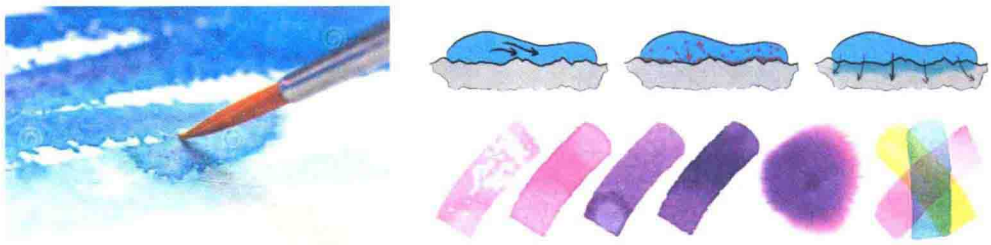


图 6.11 水彩画绘制的笔画模型（图片来自[51]）

在水彩画创作时，纸张上的液体流动状态主要受其表面粗糙程度的影响。这里可以使用简单的随机高度场 h 作为笔画模型中与纸张因素相关的参数。为了增加表现力，该高度场可以通过柏林噪声（Perlin Noise）来进一步提高凹凸的随机性。这种方式创建的高度场在各点处的高度就可以看作是纸张表面的凹凸起伏，表现出表面粗糙程度。

对于笔画模型中的其他参数，则需要考虑液体流动性对它们造成的影响，尤其是对笔画方向、颜色、浓度等参数的影响。通常借助分层流体模拟的方式设置相应的参数。这里纸张之上的液体主要包含三个层次：浅水层、附着层和扩散层。浅水层用于描述纸面之上的液体流动，附着层用于描述具有吸附/漂浮效果的纸面，扩散层则描述笔画在纸面上随液体的扩散。每一层上的笔画都是通过流体速度的变化来设置相应的参数。例如，笔画方向 $\boldsymbol{d}$ 要参考当前位置的速度方向进行设置；笔画浓度 i 的设置则要结合液体流动速度的大小以及分层的属性，这样才能够反映出笔画随液体在纸面上的扩散情况。

有了上述水彩笔画模型，绘制时就可以采用经典的 Kubelka-Munk 模型。该模型广泛用于颜料配色，特别是电子计算机配色，并用于描述颜料的遮盖力等光学性能。在各个分层上根据笔画参数进行绘制，然后通过逐层叠加就可以生成最终的水彩画绘制结果。

6.2.4 素描风格化绘制

素描是绘画的基础。广义上的素描，泛指一切单色的绘画，起源于西洋画家对实物造型能力的培养。素描按照描画材料的不同可以分成木炭素描、铅笔素描、炭精素描、钢笔素描、银笔素描、毛笔素描等形式，也就是根据产生素描笔画时所使用工具的不同而区分。这些素描通常以线条粗细、密集程度等来画出明暗效果，使其具备自然律动感。此外，素描在绘制过程中不需要顾虑物体细节的颜色。因此，笔画建模同样适用于素描风格化绘制。此时的笔画模型主要考虑形状、尺寸、浓度等笔画参数如何表示，以使得绘制结果更

能体现素描风格中的各种明暗阴影效果。

素描风格化绘制往往采用基于交互式的笔画绘制方法，需要依赖于一定数量的用户交互来完成这种风格化的绘制。这种方法的基本思想是交互地指定绘制所需的笔画模型参数，主要是方向、形状、浓度等参数的交互式设置(如图 6.12(a)所示)。例如，笔画方向 d 通过交互地指定图像上的方向场来确定每个像素对应的笔画方向。这时通常就按照从粗到细的策略将整个图像做区域划分，那么同一区域的像素被指定为同样的笔画方向。笔画形状 s 采用三次 B 样条曲线来设计，而且可以通过几个 B 样条曲线的有序且连续的组合，生成更加复杂的笔画形状。这里是充分利用了 B 样条曲线在造型方面的灵活性，用于表示素描时的各种笔画。笔画浓度 i 是通过交互涂画的方式来形成灰度图，以此表示不同位置的笔画浓度。例如，灰度值较大的涂画表示较浓的笔画，反之则是较淡的笔画。而在交互时，也要将输入图像作为参考，然后通过那些涂画过的一些采样点位置的灰度值来插值其余像素的灰度，形成符合输入图像内容的浓度分布。

基于上述笔画模型，就可以对输入的图像进行素描风格化绘制。这里在绘制时，仍是遵循分层绘制的步骤，对绘制的笔画逐层进行叠加。这样将不同尺寸的笔画相互覆盖，就能够得到最终绘制的素描风格化效果。图 6.12(b)中所示的素描风格化绘制结果能够表现出不同的明暗分布，就是通过改变笔画尺寸、浓度等参数进行绘制所生成的阴影效果，这也充分体现了素描的风格特点。

(a) B 样条定义的笔画形状　　(b) 不同笔画参数的绘制结果

图 6.12　素描风格化绘制(图片来自[52])

6.3　基于纹理合成的绘制

纹理泛指物体表面上的纹路或线条，它在视觉上是呈现相似而连续的统计分布。在传统计算机图形学中，纹理是指物体表面的图形表示，例如第 5 章真实感绘制中介绍的纹理映射技术。而在传统图像处理中，纹理是指具有自身重复性的图像。构成纹理的基本单元称为纹素。纹理合成，是指基于给定的局部范围内小区域纹理样本作为纹素，直接生成更大范围的纹理。这里往往需要按照图像内容的几何、颜色等结构分布，将小区域纹理拼合生成更大区域的纹理图像。因此，纹理合成的过程其实和一些非真实感绘制过程非常相似。这就启发了采用纹理合成进行非真实感绘制的方法。

接下来，介绍两种典型的基于纹理合成的油画风格化绘制方法：基于风格类比和基于笔画的纹理合成绘制方法。

6.3.1 基于风格类比的纹理合成绘制方法

这种方法与前面的笔画风格迁移类似，但区别是迁移的对象不同。基于风格类比的纹理合成绘制方法是将输入的两幅模板图像之间的类比关系迁移至目标图像，然后借助纹理合成的思路生成目标图像的非真实感绘制结果。因此，这里的风格迁移，就是要求图像之间的类比关系的一致。如图 6.13 所示，给定 A 和 A' 作为一对模板图像，其中 A' 是对 A 施加风格化处理后的图像。那么，基于风格类比的纹理合成绘制就是要在给定目标图像 B 后，合成一个“类似”的新图像 B'，使得 B 和 B' 之间的关联性类似于 A 和 A' 之间的关联性。

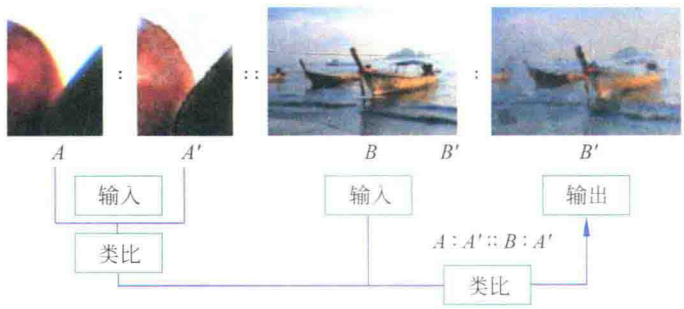


图 6.13 基于风格类比的纹理合成绘制(图片来自[53])

这个算法流程主要包括图像匹配和风格化图像的纹理合成两个阶段。在图像匹配阶段，需要寻找 A 中每个像素 $\boldsymbol{p}$ 及其周围的相邻像素对应于 A' 中相同的像素 $\boldsymbol{p}$。在给定 B 作为输入的目标图像后，B' 和 B 之间的像素对应关系需要具有 A' 和 A 的这种匹配效果。这也是进行风格类比的关键步骤。实际处理时，往往通过建立多尺度的图像序列来提取简单的风格特征，然后按照对应序列等级进行图像分块比较，建立不同尺度下的匹配关系。

在纹理合成时，则采用经典的扫描线策略，根据匹配的分块逐行来合成当前位置像素的颜色值。例如，为了合成 B' 中 $\boldsymbol{q}$ 处的像素值，需要在 A 图像中搜索与 B 图像中对应像素最相似的匹配像素 $\boldsymbol{p}$。这样就需要根据模板图像对应关系找到 A' 中的像素及其邻域合成 $\boldsymbol{q}$ 处像素值(如图 6.14 所示)。为此，采用近似最近邻搜索(ANN)策略，建立图像对

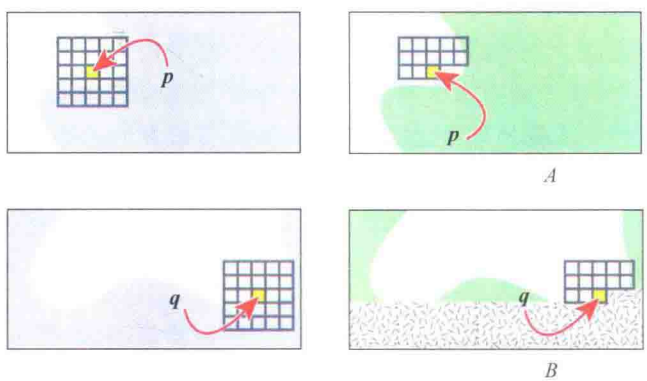


图 6.14 格类比时的匹配像素搜索

之间的匹配关系。这里既要考虑对应像素及邻域的相似性，也要考虑相邻像素匹配位置的一致性，从而得到图像对之间的最优匹配位置。因此，由于模板图像对之间潜在的映射关系，这种匹配的块在合成时往往能够取得较高的一致性，生成的新图像在继承风格的同时，整体自然性也较高。

6.3.2 基于笔画的纹理合成绘制方法

这种方法是将艺术作品图像局部的笔画分布作为一种纹理，称为笔画纹理。它实际上是用包含了笔画模型的局部区域作为纹素。那么，如图 6.15 所示，由笔画覆盖所形成的绘制效果就可以仿照图像纹理合成的方式进行创作。这种笔画纹理既可以作为一种直接显式的笔画模型，称为样本笔画纹理；也可以作为一种间接隐式的笔画绘制效果的样本，称为样本图像纹理，然后再按照输入图像的内容进行绘制。那么在实际的非真实感绘制过程中，按照笔画纹理选取的差异，就可以分别采用基于样本笔画纹理绘制方法，或者采用基于样本图像纹理绘制方法。

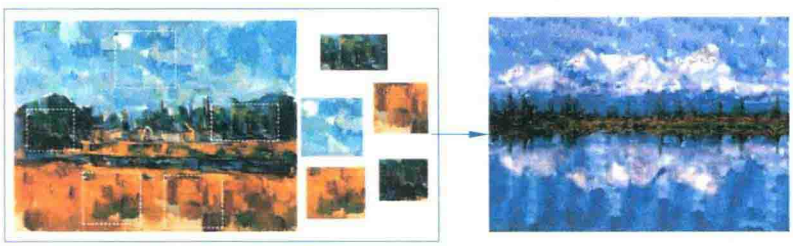


图 6.15 笔画纹理合成的风格化结果(图片来自[54])

基于样本笔画纹理的绘制，同样需要借助一些艺术家手工绘制的笔画作为样本。这些样本笔画呈现不同的尺寸、方向、形状等参数，以此提高对输入图像非真实感绘制时的笔画效果。此外，通过预设的高度纹理图和透明纹理图，分别去定义笔画颜色、亮度以及浓度等参数。那么在绘制时，以输入图像作为目标，确定样本笔画的先后顺序，并按照选择的颜色、亮度、浓度等参数进行设置，通过纹理合成的方式生成绘制后的图像。这种方法得到的风格化结果具有很强的笔触感，但是需要借助一定规模的样本笔画库才能达到理想的绘制效果。

基于样本图像纹理的绘制，选择指定风格图像的局部区域作为样本笔画(如图 6.15 所示)，而非艺术家专门创作的笔画。而在绘制时，则需要借助输入图像的方向场指导纹理合成时的笔画覆盖方式。具体来讲，首先需要从给定的模板油画中提取样本笔画，也就是样本图像纹理。这里通常直接采用手工选择的方式，由用户通过勾画矩形框来挑选模板中笔画明显区域作为样本。显而易见，选取的区域应该具有明显且均匀的纹理，这样才能更清晰地反映笔画的风格特点。

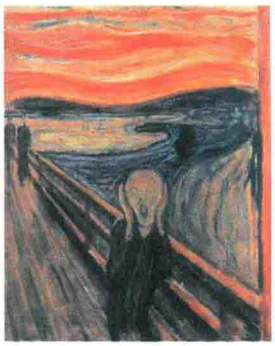
为了适用于不同图像内容的绘制，需要再对挑选的样本采用金字塔高斯滤波生成不同尺度下的序列，并进一步对其做 360°旋转采样。这样就可以生成不同方向的样本区域集合。这种方式构建的样本区域就能够适用于目标图像中不同的物体及区域，从而达到更好的绘制效果。然后，对目标图像使用均值漂移等聚类算法进行区域分割，并计算每个

区域的中轴曲线。这样每个区域具有较均匀的颜色和梯度方向分布。特别地，位于中轴上的像素点的梯度方向被选为该点处的中轴切线方向，那么区域中其他像素的方向在设置时就需要和最靠近该像素点处的中轴上像素点的方向一致。这样才能够生成符合图像内容的方向场。最后，根据目标图像上每一块区域对应的方向场和颜色分布，选取符合当前方向和颜色的样本图像进行纹理合成，从而获得符合模板图像风格的绘制结果。

在纹理合成时，可以将块重叠拼合时的逐像素误差最小作为优化目标。这种合成方法依次将样本区域在目标图像上覆盖，而且要在相邻区域设置一定范围的重叠区域。然后，在该重叠区域通过图割优化寻找像素误差最小的分割线，将相邻样本区域拼合。这样就得到了基于样本区域的纹理合成结果。

例题 6-2　基于笔画纹理合成的绘制。

问题：给定如下图所示的 A(左图)和 B(右图)两幅图，其中 A 图是模板油画，B 图是待绘制的自然风景图像。试通过交互方式选择 A 图中的纹理块，合成 B 图对应的风格化绘制结果。



A

B

解答：从模板油画图像 A 中提取的纹理块记为 T_{patch}，宽度和高度分别是 T_w 和 T_h。输入图像 B 的宽度和高度分别是 w 和 h，而在合成过程中被纹理块覆盖的区域记为 I_{patch}，相邻块重叠区域的范围记为 p。那么，通过纹理块合成进行风格化绘制的伪代码如下所示。

```
function texSynDraw (Ipatch, Tpatch, w, h)
    for y ← 0 to h-1 do
        for x ← 0 to w-1 do
              patch (x, y) ← min (Ipatch(x, y), Tpatch)
              x ← x +patch_width-p
        end for
        y ← y +patch_height-p
    end for
    for y ← 0 to h -patch_height +p -1 do
        for x ← 0 to w -patch_width +p-1 do
            horizontal_min_error_path(x, y) ← get_shortest_path(patch(x, y),
                                        patch(x+patch_width-p, y))
```

```
            row_patch(y) ← row_patch(y) +systhesis_horizontal_patch(patch(x, y),
                    patch(x +patch_width-p, y), horizontal_min_error_path(x, y))
            x ← x +patch_width-p
        end for
        y ← y +patch_height-p
    end for
    for y ← 0 to h-patch_height +p-1 do
        vertical_min_error_path(y) ← get_shortest_path(row(y),
                                row(y +patch_height-p))
        transfer_image ← transfer_image +systhesis_vertical_patch(row(y),
                row(x +patch_height-p), vertical_min_error_path(y))
        y ← y +patch_height-p
    end for
end function
```

□

6.4 基于图像滤波的绘制

滤波一词起源于通信理论，它是从含有干扰的接收信号中提取有用信号的一种技术，也就是将信号中特定波段频率滤除的操作。图像滤波，是将像素强度作为信号，在尽量保留细节特征的前提下，对目标图像的噪声进行抑制，达到平滑像素强度的效果。常见的滤波方法有高斯滤波、中值滤波、双边滤波等。

卡通作为一种艺术形式最早起源于欧洲，常常使用抽象化的符号、颜色、亮度和阴影等手法体现夸张的视觉效果。卡通图像大多线条简单、形式灵活，广泛应用于动画和电影，深受不同年龄段人群的喜爱。卡通风格化，就是将输入图像转化为具有明显轮廓线、色彩对比鲜明的图像，以此体现卡通风格效果。因此，图像滤波非常适合于卡通风格化绘制。接下来，介绍若干典型的基于图像滤波的卡通风格化绘制技术。

6.4.1 基于流体场的双边滤波绘制

双边滤波是一种非线性的滤波方法。它结合图像像素的空间邻近度和颜色值相似度对输入图像进行滤波操作，以达到保持边缘、降噪平滑的目的。从滤波的角度来看，卡通图像可以看作是显著线条和平滑色块叠加的结果。借助双边滤波，可以对图像颜色进行边缘保持的区域平滑，形成颜色均匀的色块。此外，对于一般的自然图像，物体边缘往往形成类似流体场的分布。因此，也可以从图像边缘定义的流体场提取显著线条。在此基础上，结合双边滤波色块和流体场显著线条，将两者做适当的叠加，就可以把输入图像转化成具有卡通风格的图像。

在基于流体场的双边滤波绘制过程中，第一步需要从输入的图像中提取具有显著特征的线条结构。这里采用基本的图像边缘检测算子，例如 Sobel 算子，得到边缘响应分布图。这其中就包含了卡通风格化后需要保留下来的显著线条。边缘流体场则通过统计局部窗口主要方向的累计效应进行计算，形成图像上的边缘切向流。借助该流体场，就可以

摒除边缘响应分布图中不重要的边缘，而保留用于卡通风格效果的显著线条。

数学上，在区域 Ω_μ 内的点 $\boldsymbol{x}$ 处的边缘切向流是通过局部区域加权的边缘积分表示，具体可写为：

$$\boldsymbol{t}(\boldsymbol{x}) = \frac{1}{k}\iint_{\Omega_\mu}\varphi(\boldsymbol{x},\boldsymbol{y})\boldsymbol{t}(\boldsymbol{y})w_s(\boldsymbol{x},\boldsymbol{y})w_m(\boldsymbol{x},\boldsymbol{y})w_d(\boldsymbol{x},\boldsymbol{y})\mathrm{d}x\mathrm{d}y \tag{6.1}$$

其中，$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 是流体场覆盖区域 Ω_μ 内的点。如果 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 处切向一致，则 $\varphi(\boldsymbol{x}-\boldsymbol{y})=1$，反之，$\varphi(\boldsymbol{x}-\boldsymbol{y})=-1$；$w_s$ 是箱滤波算子，指定局部积分区域大小；w_m 衡量邻域内梯度大小的差异；w_d 表示邻域内梯度方向的差异。然后，通过高斯函数差分(DoG)从边缘切向流中提取出那些显著线条。这里主要是通过对两个相邻高斯尺度空间的图像相减，得到 DoG 的响应值图像作为提取的显著线条位置。

第二步，将显著线条以外的图像区域，采用双边滤波进行平滑处理。在滤波时，沿着显著线条，并根据像素与线条距离来进行自适应的平滑处理。这样处理的好处是可以得到符合局部颜色分布的平滑色块，并以此作为卡通图像的色块。

在完成上述两步操作后，就可以将前面提取到的显著线条叠加到平滑色块上，生成卡通风格化处理的效果。如图 6.16 所示，可以看到通过流体场和双边滤波的处理，能够很好地突出图像中物体的轮廓，同时色块区域也具有较均匀的颜色分布，从而实现了卡通风格化效果。

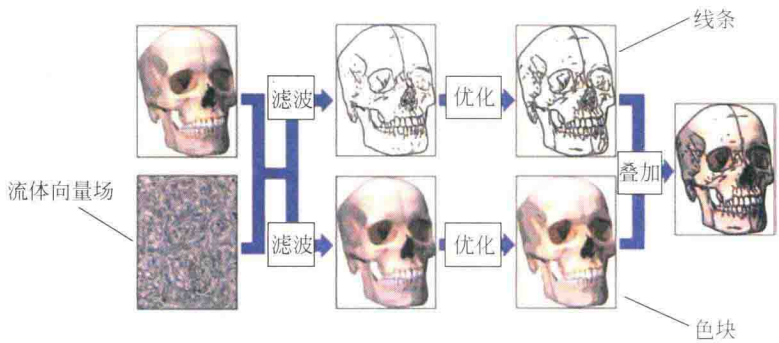


图 6.16 基于流体场的卡通绘制(图片来自[55])

6.4.2 基于亮度的双边滤波绘制

虽然基于流体场的方法可以有效地生成卡通图像，但是平滑色块所展现的颜色又在一定程度上缺乏区域内的空间变化，从而造成卡通效果比较单调、不够生动，无法满足一些卡通形象设计的需求。这是由于在使用双边滤波时，仅仅是对颜色进行处理造成的。那么，改进的思路是可以再结合亮度通道的平滑处理，实现卡通绘制时色块区域内的变化，进一步增强卡通绘制的立体感。

这种思路可以采用迭代的双边滤波来具体实现。如图 6.17 所示，首先对输入图像进行颜色空间变换。将输入图像从 RGB 空间映射到 Lab 空间，其中 L 通道就包含了图像亮度信息。然后使用迭代双边滤波对 L 通道分量进行处理。这样在增大图像对比度的时候，也保留了区域内明显的亮度变化。接着，通过对 a 和 b 两个通道分量的锐化处理，

提取输入图像中的显著边缘，并与亮度通道双边滤波结果进行叠加。最后，将处理后的图像从 Lab 空间映射回 RGB 空间，就生成了具有卡通效果的图像。

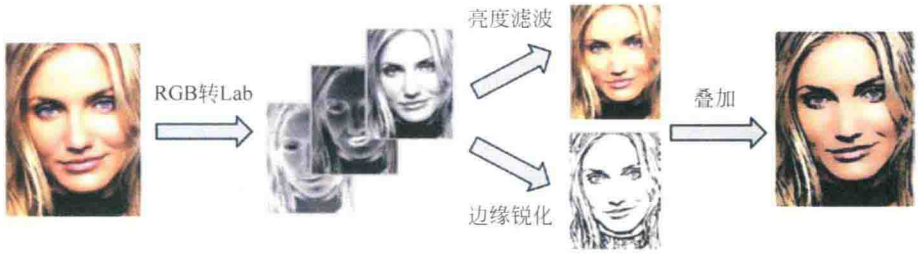


图 6.17 基于亮度滤波的卡通绘制(图片来自[56])

从图 6.17 所示的结果可以看出，由于额外增加了对亮度通道分量的处理，这使得在原本颜色均匀的同一区域内部，其亮度分布也呈现出了一定的变化。因此，通过这种方法生成的卡通风格化结果更加具有立体感，由此设计的卡通形象也会更加生动。

6.4.3 基于纹理/结构分层的滤波绘制

传统双边滤波难以处理图像中对比度较低区域的颜色平滑，导致滤波后会丢失部分重要的细节信息。这是由于尺度的差异影响了对比度的区分。另一种可使用的滤波思路是将图像分解为纹理层和结构层，然后在结构层上对图像进行平滑处理。这样处理的好处在于能够更好地保留输入图像中视觉显著的边缘结构，而且这种结构受尺度的影响较小。这种方式也有助于提高卡通风格化的效果。

从认知心理学角度来讲，视觉的显著性可以通过视觉信号的各向异向、非周期性、局部方向性等特征进行描述和度量。因此，可以从这一视觉认知机理出发，将输入图像自动地分解为纹理层和结构层。这种分层方式本质上是利用边缘的各向异性、非周期性和局部方向性度量来区分图像中结构显著性的强弱，并以此构建图像的分层表示。在此基础上，通过输入图像对该度量的响应，建立图像边缘的显著结构表达，从而获取图像的结构层。一般来讲，如图 6.18 所示，结构层保留了显著边缘信息，而纹理层则对应于纹理区域。这样就可以分别利用两层的信息进行风格化处理。

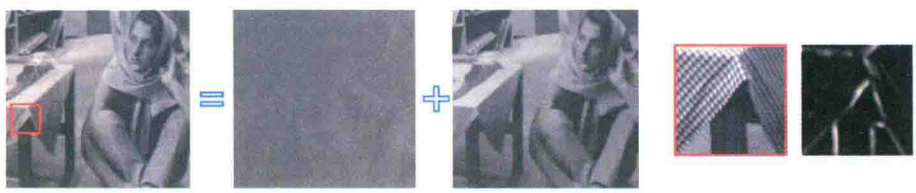


图 6.18 基于边缘显著性的图像分层和显著性度量(图片来自[57])

在对图像进行特征描述时，各向异性采用图像梯度空间的结构张量作为梯度向量的度量函数。这样，对于输入图像，可以计算每一个像素点 $\boldsymbol{p}$ 所对应的二阶张量矩阵 S_p。数学上，针对该像素点处各项异性的度量函数可以定义为

$$A_p = (\lambda_{1,p} - \lambda_{2,p})/(\lambda_{1,p} + \lambda_{2,p}) \tag{6.2}$$

其中，$\lambda_{1,p} \geqslant \lambda_{2,p}$ 是梯度空间的张量矩阵奇异值。

非周期性可以采用基于窗口梯度变差分布的度量函数来计算，具体公式是

$$L_p = \left| \sum_{q \in N(p)} w_{p,q} \cdot \nabla_x I_q \right| + \left| \sum_{q \in N(p)} w_{p,q} \cdot \nabla_y I_q \right| \tag{6.3}$$

其中，∇ 是梯度算子。在一定窗口范围内重复的图像颜色变化，其 x 方向和 y 方向的梯度分量累积就会彼此抵消，对于图像中周期性出现的纹理区域而言，该度量函数的取值就会趋近于零，因此可以用于度量局部范围内的非周期性。

局部方向性度量可以通过统计局部方向分布的相关性来实现，并以此判断在局部区域内是否有显著方向的特征分布，具体公式是

$$D_p = \sum_{q \in N(p)} \boldsymbol{A}_q \cdot \langle \bar{\xi}_q, \bar{\xi}_p \rangle \Big/ \sum_{q \in N(p)} \boldsymbol{A}_q \tag{6.4}$$

其中，$\bar{\xi}_p$ 和 $\bar{\xi}_q$ 是张量矩阵最小特征值对应的特征向量，反映了局部范围内边缘走向的方向。那么，最终的边缘显著性度量就可以写为 $M_p = A_p \cdot L_p \cdot D_p$，实质上反映了图像对上述三种度量的共同响应。

基于上述边缘显著性作为约束，就可以通过经验模式分解得到最大值和最小极值包络，从而对结构层和纹理层实施分离。经验模式分解是一种依据数据自身尺度特征进行分解的滤波方法，在数学上能够将复杂信号分解为有限个本征函数。通过这种方式得到的结构层就可以用于对图像的平滑。在此基础上，通过将分离出的边缘结构强化，再叠加到平滑图像上进行绘制，就可以生成理想的具有卡通风格效果的图像（如图 6.19 所示）。

图 6.19 基于纹理/结构分层滤波的卡通绘制结果（图片来自[57]）

6.5 视频非真实感绘制

图像非真实感绘制是将输入的单幅图像转化成具有一定艺术风格的图像。而视频非真实感绘制则是图像非真实感绘制的推广，将视频帧形成的序列进行风格化处理，输出具有特定艺术感的新视频。然而，视频记录的内容通常包含场景或物体的运动。因此，与单幅图像的非真实感绘制相比，视频绘制的难点主要在于如何保持各帧绘制时的时空连续性，从而能够达到相邻帧之间视觉连续和一致的绘制效果。这里的一致性具体表现为帧间笔画建模的一致性和绘制过程的一致性，是视频绘制需要解决的重要问题。

视频帧间的运动可以采用光流、视频体等不同的形式表示。这些表示形式也直接影

响了绘制方法的使用。光流，是描述空间中运动的物体在相邻帧上的像素瞬时速度。数学上，光流通常采用对应于每一个像素（或特征点）$\boldsymbol{p}_i$ 的位移向量表示，记为(u_i, v_i)。这里 u_i 和 v_i 是位移向量在 x 方向和 y 方向上对应的分量。它反映了相邻帧间像素级的对应关系。视频体则是一种视频内容在时空维度连续变化的表示形式。它将每一帧按时间维度的先后顺序堆叠，形成(x_i^t, y_i^t, t)所表示的体数据，这里(x_i^t, y_i^t)是第 t 帧上第 i 个像素的位置。根据不同的视频运动形式，就可以在对视频进行逐帧笔画绘制的同时，有效地保持帧间绘制效果的一致性，实现视频的非真实感绘制。

6.5.1 基于帧间光流的笔画绘制

利用光流产生的位移向量，可以在相邻帧之间传递笔画参数。因此，视频绘制时的笔画建模仍旧可以采用前面介绍的针对图像绘制的各种笔画模型，也就是需要根据风格特点设置相应的笔画尺寸、形状、方向、颜色、浓度等参数，进而绘制视频帧序列。接下来以视频的油画风格化为例，介绍基于帧间光流的笔画绘制方法。

对于单帧图像，为了实现更好的油画风格，可以采用基于笔画距离的衰减程度来表示笔画浓度，从而使笔画呈现出从中间向两边递减的浓淡分布。在此基础上，为了保持物体的边缘结构，采用笔画裁剪方法，将过长的笔画进行缩短，使绘制后的笔画不得越过物体本身的边缘。这样就可以避免绘制过程中物体本身结构的破坏，使得绘制的结果能够如实反映视频每帧的内容（如图 6.20(a)所示）。这里需要先对原视频帧进行高斯模糊，然后执行 Sobel 边缘检测，并对笔画进行裁剪。如果不考虑该帧图像中物体原本的边缘，就会使得笔画绘制后的结果的边缘线出现凌乱的锯齿状，降低了风格化效果。

下一步，如图 6.20(b)所示，通过帧间光流来追踪相邻帧间对应像素的移动方向，使笔画从当前一帧移动到下一帧。这样就可以确保笔画的一致性。进一步，通过改变笔画浓度等参数就可以调整绘制后的风格特点。

(a) 笔画裁剪　　(b) 光流跟踪

图 6.20 基于光流的视频帧绘制（图片来自[58]）

视频场景包含不同的物体前景和背景，往往各自具有不同的运动属性。根据每一帧上光流的分布，就可以对笔画进行运动分层的帧间传播。和前面直接在两帧之间通过光流传播笔画不同，这种方法首先通过光流进行一个简单的运动分层；然后在不同分层上根据实际的运动属性，传递相应的笔画，并在每一个运动层上进行笔画绘制；最后将绘制完成的各层重新投射到每一帧，并经过融合得到最终的绘制结果（如图 6.20 所示）。这里，

视频其实是被当作由一个背景层和所有帧中的若干前景对象层所组成的。通过在每一层上建立符合其运动属性的笔画模型来实现帧间一致的风格化绘制。

为了实现准确的运动分层，需要通过关键帧上交互划分的方式，来引导其他剩余帧上的分层。在关键帧上，通过鼠标涂画指定分层的层数，以及相应分层的种子区域。这些种子区域就传达了用户对于运动分层的设想。同时，将涂画覆盖区域作为约束，采用图割优化等方法把关键帧划分为不同区域。其中把光流作为直接的约束，以获得具有相应运动属性的运动分层。然后，利用传统图像笔画绘制方法绘制关键帧。

在接下来的帧间笔画传播时，首先需要将关键帧的运动分层结果传递到其他帧。因为关键帧上的种子区域提供了关于分层的主要信息，所以将关键帧的种子区域根据帧间光流移动到相邻帧，然后再按照关键帧上的分层方法进行相应的区域分割，得到该帧上的运动分层。而在进行绘制时，则需要将背景层和前景层分别进行绘制。对于背景层，由于每一帧中的背景都是一个静态场景随着相机移动得到的，因此可以利用背景图拼接而成的全景图来提高视频绘制的时空一致性。这里，通过对齐视频中所有帧上背景层来生成全景背景图。这个过程实质上是多幅图像的对齐拼接，将在第 7 章中介绍相应的方法。在此基础上，按照指定参数的笔画绘制全景背景图。因为该全景图是通过对齐操作生成的，所以可利用逆操作将绘制的全景图再映射回每一帧，由此得到每一帧上绘制的背景层。

而在绘制前景时，不同帧上对象层所对应的笔画主要通过尺寸、位置、方向等属性来定义，但是相邻帧之间需要满足时空一致性。这里通过光流诱导的几何变换，将第 k 帧中的笔画属性变换到第 $k+1$ 帧，然后利用笔画模型进行绘制。这样可以保持较好的帧间连续性。最后，将绘制的前景层和背景层进行融合，形成完整的笔画绘制（如图 6.21 所示）。

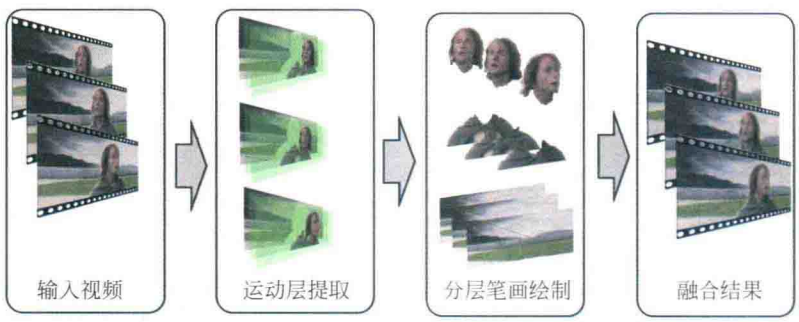


图 6.21 基于运动分层的视频帧绘制（图片来自[59]）

6.5.2 基于视频体的笔画绘制

这种方法的主要思想是把一段输入的视频当作一个时空立方体，根据帧间内容进行一致的笔画建模。而在绘制时，往往需要用户交互地对物体进行分割，然后根据对象的不同运动属性自动地进行笔画绘制。

这里对视频体中的物体对象进行合理的分割是关键。首先借助传统的分割方法，如 Edsion 算法，对每一帧进行过分割，形成颜色一致的分块集合。然后对于关键帧上分割

错误的区域，交互式地进行手工修正。这个过程称为语义连接，也就是将属于同一物体对象的过分割区域合并。此外，对于阴影或噪声等产生的分割错误，也需要交互地进行修正，称为物理连接。通过这两步操作后，就可以得到关键帧正确的对象分割（如图 6.22 所示）。

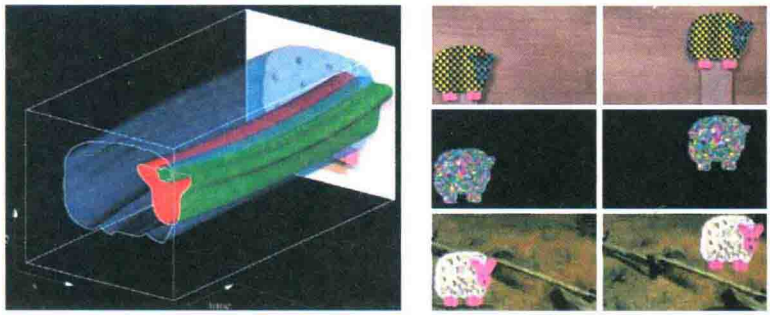


图 6.22 基于视频体的视频帧绘制（图片来自[60]）

接下来，以关键帧的分割作为约束，通过基于体的图割优化策略对该视频体包含的帧进行对象分割。在得到逐帧分割结果后，利用参数随时间一致变化的笔画模型在空间进行不同对象的绘制。最后，将所有绘制的对象融合，就可以得到视频对应的绘制结果。

6.6 基于深度学习的绘制

归根结底，艺术创作是一种包含了人类主观意识的活动。因此，在非真实感绘制时，对图像进行更高级的特征分析与建模，将能够大大提升绘制后的风格化效果。这从前面所介绍的几种自动笔画绘制方法也可以看出。然而，从输入的图像提取高级特征，一直是计算机图形学和计算机视觉长期难以逾越的屏障。近些年来，以卷积神经网络、生成对抗网络等为代表的深度学习方法在计算机图形学和计算机视觉的许多任务中都得到了广泛应用，取得了显著的成果。这也为非真实感绘制提供了新的途径。

深度学习是通过多层神经网络上的学习算法来构建的框架。这种框架能够从输入的大量数据获取分层次的特征信息。较低层的网络可以检测初级的图像特征，如边缘、轮廓、形状等。通过底层特征在多个层级的组合，最终在较高层网络就能够识别一些高级的、语义的特征。因此，借助深度学习进行非真实感绘制，能够通过挖掘高级语义信息来更有效地提高艺术风格化效果。

接下来，简单介绍两种典型的基于深度学习的图像风格迁移算法：基于卷积神经网络的风格迁移和基于生成对抗网络的风格迁移。

6.6.1 基于卷积神经网络的风格迁移

卷积神经网络（Convolutional Neural Networks，CNN）是一种包含卷积计算且具有深度结构的多层神经网络，其中层数越多也意味着深度越深。卷积神经网络通过模仿局部感受野的卷积计算来对不同级别的特征进行区分和表示，是一种典型的监督学习方法。

这样，如果将卷积神经网络用于输入的自然图像和模板油画图像的表示，那么就可以显式地利用更深层次的高级特征，指导从模板图像到输入图像的风格迁移。如图 6.23 所示，输入图像通过卷积神经网络在每一层上响应的特征，获得了场景内容的形式化表示，而模板图像通过卷积神经网络每一层响应的特征，也获得了风格的表示。那么，待求解的迁移图像则需要在每一层上的响应既要满足与输入图像内容相似，又要达到与模板图像风格相近的效果。

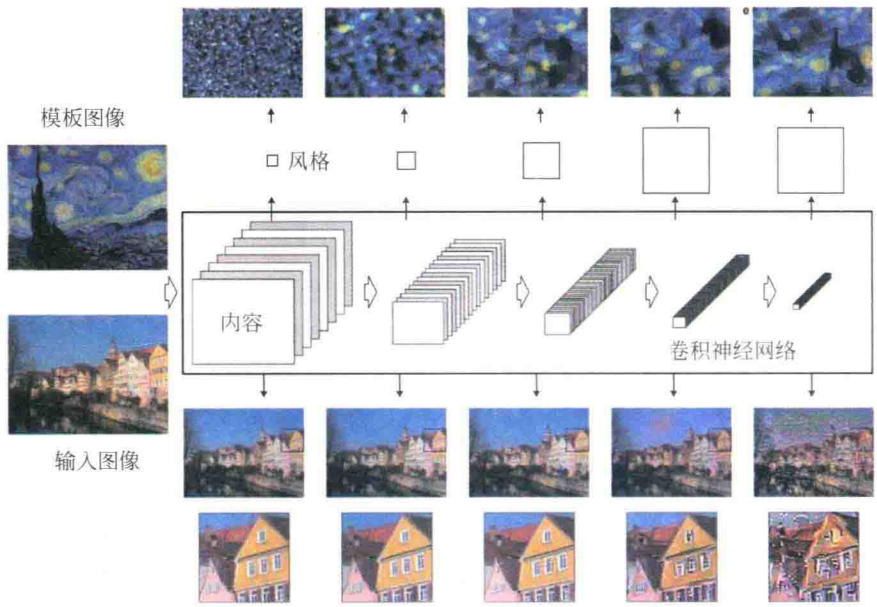


图 6.23 通过卷积神经网络提取输入图像的内容和模板图像的风格(图片来自[62])

假设 $\vec{p}$ 是输入的自然图像，$\vec{a}$ 是模板油画图像，通过卷积神经网络提取的特征分别作为内容和风格的表示。那么，待求解的具有 $\vec{a}$ 风格的输出图像 $\vec{x}$ 应该使得以下损失函数能够达到最优：

$$\mathscr{L}_{\text{total}}(\vec{p},\vec{a},\vec{x}) = \alpha\mathscr{L}_{\text{content}}(\vec{p},\vec{x}) + \beta\mathscr{L}_{\text{style}}(\vec{a},\vec{x}) \tag{6.5}$$

其中，$\mathscr{L}_{\text{content}}$ 和 $\mathscr{L}_{\text{style}}$ 分别对应输入图像内容和模板图像风格的损失函数。这里 α 和 β 是预设的权因子，用于控制输出图像和输入图像、模板图像之间的相似程度。进一步，损失函数 $\mathscr{L}_{\text{content}}$ 描述了使用该卷积神经网络提取的特征来重建输入图像时的误差代价，其中对应于第 l 层网络可定义为

$$\mathscr{L}_{\text{content}}(\vec{p},\vec{x},l) = \sum_{i,j}(F_{ij}^{l} - C_{ij}^{l})^2/2 \tag{6.6}$$

这是逐像素定义的输入图像 $\vec{p}$(特征表示为 F^l)和待求解图像 $\vec{a}$(特征表示为 C^l)重建时的误差。另一方面，损失函数 $\mathscr{L}_{\text{style}}$ 则描述了重建模板图像时的误差，对应于第 l 层网络可定义为

$$\mathscr{L}_{\text{style}}(\vec{a},\vec{x},l) = \frac{1}{4N_l^2M_l^2}\sum_{i,j}(G_{ij}^{l} - S_{ij}^{l})^2 \tag{6.7}$$

其中，N_l 是第 l 层提取的特征数目，M_l 是特征维数，S_{ij}^{l} 是通过卷积神经网络提取的模板

图像特征构成的 Gram 矩阵，G_{ij}^{l} 是待求解图像对应的 Gram 矩阵。

这里的卷积神经网络可以使用基于物体识别任务训练过的 VGG 等网络，直接提取输入图像作为内容的表示，提取模板图像特征作为风格的表示。同时，采用白噪声图像作为待求解的风格化图像的初值。这其实也是充分利用了卷积神经网络对于图像不同层次特征的提取和表示能力，从而将内容和风格进行有效结合。在此基础上，通过经典的梯度下降法优化公式(6.5)所定义的目标函数，使其总体损失函数取值最小。这种方式得到的最优解就对应于将任意输入图像转换成指定模板图像风格的效果，从而实现了模板图像到输入图像的风格迁移(如图 6.24 所示)。显而易见，基于卷积神经网络的风格迁移能够处理不同的模板图像，将输入图像同时转化成各种不同风格的图像。

图 6.24　基于卷积神经网络的风格迁移结果(图片来自[62])

6.6.2　基于生成对抗网络的风格迁移

生成对抗网络(Generative Adversarial Networks，GAN)也是一种深度学习模型，被认为是无监督学习最具前景的方法之一。生成对抗网络包含两部分模型：生成模型(Generator)和判别模型(Discriminator)。在优化过程中，通过这两部分的博弈学习产生输出，从而可以看作是一种典型的无监督深度学习方法。这种神经网络在训练时不需要严格标记图像对作为样本，而是通过优化生成器的损失函数和判别器的损失函数，最终获得符合样本统计分布的生成数据，并将其作为优化结果。如图 6.25 所示，这样就可以将某一风格的模板图像作为样本集合，而利用生成器将输入图像映射为符合样本分布的图像，同时通过判别器给出的损失函数不断更新生成器的网络参数，使其能够输出更好的生成图像。最终，待求解的迁移图像就可以通过平衡状态下的生成器得到风格化结果。

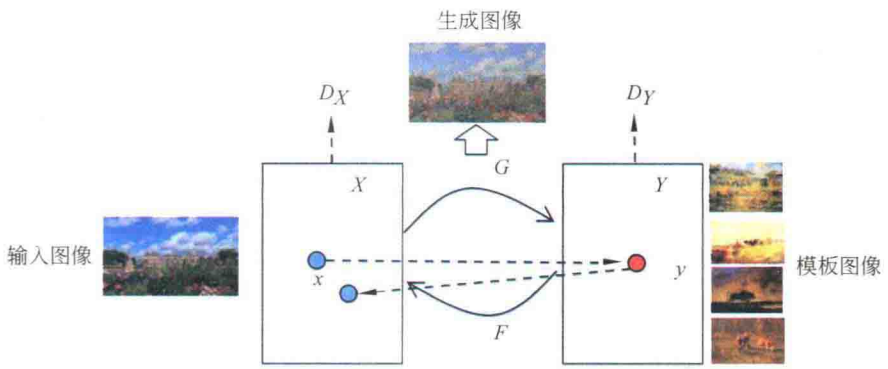


图 6.25　通过生成对抗网络产生符合某一风格模板图像集合样本分布的风格化图像

具体来讲，假设 X 对应于输入图像，Y 对应于模板图像，那么映射 $G:X\rightarrow Y$ 作为生成函数就会将输入图像转化为风格化图像。而其反向映射 $F:Y\rightarrow X$ 则能够在无法提供标记对应样本的时候，使得生成图像符合输入图像的内容。因此，映射 F 和 G 描述图像内容和风格。此外，D_X 和 D_Y 是对抗判别器，分别用来判定 $\{x\}$ 和 $\{F(y)\}$ 的差别，以及 $\{y\}$ 和 $\{G(x)\}$ 的差别。在此基础上，该网络对应的损失函数定义为

$$\mathcal{L}(G,F,D_X,D_Y)=\mathcal{L}_{\mathrm{GAN}}(G,D_Y,X,Y)+\mathcal{L}_{\mathrm{GAN}}(F,D_X,Y,X)+\lambda\mathcal{L}_{\mathrm{cyc}}(G,F) \quad (6.8)$$

其中包含了对抗生成网络两个经典的的损失函数 $\mathcal{L}_{\mathrm{GAN}}$ 和一个循环损失函数 $\mathcal{L}_{\mathrm{cyc}}$。这里需要注意，由于采用了映射 F 产生图像内容，使得在训练时可以使用不完全配对的输入图像和模板图像作为样本，从而更好地适用于风格迁移。此外，循环损失函数可以有效阻止通过该网络学习得到的映射 G 和 F 彼此产生冲突，引入无效的生成图像。这样就可以将任意的输入图像最终转化为不同风格的输出图像。例如图 6.26 所示的结果，同一个输入图像就被转化为具有莫奈、梵高、塞尚等艺术家作品风格的多幅油画图像，同时也能够转化为浮世绘风格。

图 6.26 基于生成式对抗网络的风格迁移结果(图片来自[63])

6.7 小 结

非真实感绘制打破了传统图形学绘制时一味追求真实性的发展，为计算机图形学注入了新的活力。自从 1997 年 SIGGRAPH 开始将非真实感绘制作为独立的会议单元以来，非真实感绘制技术进入了一个稳步发展的时期。笔画建模、纹理合成、图像滤波等技术广泛应用于不同艺术风格的非真实感绘制，并且从图像扩展到视频。

非真实感绘制更为强调视觉感知的效果，因而需要语义层面的分析与处理。随着以深度学习为基础的人工智能技术的强势兴起，传统方法无法处理的语义信息提取等问题得到了较好解决，也为非真实感绘制提供了新的发展机遇。

思考题

6.1 什么是非真实感绘制？它和真实感绘制的区别是什么？

6.2 使用笔画建模的艺术风格有哪些？不同风格笔画模型的差别是什么？

6.3 相比于油画和素描风格化绘制，水彩风格化绘制需要考虑哪些额外因素？

6.4 如何将图像纹理合成应用于非真实感绘制？

6.5 如何将图像滤波应用于非真实感绘制？

6.6 图像非真实感绘制技术推广到视频时的难点是什么？常用的解决手段有哪些？

6.7 目前流行的深度学习方法用于非真实感绘制时有哪些优势和不足？

第 7 章 基于图形的影像处理

影像是人类获取视觉信息的重要媒介，而图像、视频、图形又是展示影像内容的最重要形式。这里的图像主要指位图图像，它以像素为单位记录颜色信息。例如，常见的 24b 的 RGB 彩色图像，每个像素实际上包含了 3 个通道，分别记录红、绿、蓝三种单元色的颜色强度信息。视频是由连续的静止图像形成的帧序列，记录动态变化的视觉信息。图形则是指矢量图，以图元作为基本单位记录影像颜色、形状等视觉信息。传统的图像/视频等影像处理大多是以像素为基本元素，通过建立合理的数学或物理模型进行计算。图形蕴含了更丰富的几何信息，采用基于图形的影像处理，能够更有效地提升图像/视频处理的效果和效率。

本章介绍影像处理中图像和视频的一些图形表示方法，以及图形学技术在影像处理中的应用，重点讲述如何使用图形学方法解决传统影像处理中的问题，具体内容包括影像抠图、影像缩放、影像融合、影像拼接和影像编辑等。

7.1 影像抠图

图像/视频抠图与图像/视频分割紧密相关，都是指将图像或视频画面划分为多个区域，而且一般要求不同区域之间没有重叠。数学上，抠图结果采用掩模图(Mask Artwork)表示。掩模图记录了每个像素的抠图值，也就是对每个像素 $\boldsymbol{p}$ 赋予 0 到 1 之间的数值 α_p。其中，$\alpha_p=0$ 表示该像素属于明确的背景，$\alpha_p=1$ 表示该像素属于明确的前景，而取值介于两者之间的 α_p 则对应于前景和背景混合而得的像素。因此，图形学里面的抠图实际上强调的是一种软分割，这与图像/视频的硬分割是截然不同的。这里，硬分割是指赋予每一个像素的数值只能是 0(背景)或者 1(前景)，而不存在介于两者之间的数值。换言之，硬分割后的像素或者被划分到前景，或者被划分到背景。而对于软分割而言，还存在一些无法被完全划分到前景或背景的像素。

如图 7.1 所示，图像/视频抠图的结果 α_p 表示了某种意义下前景和背景进行混合时的像素透明度。因此，对图像 $\boldsymbol{I}$ 进行抠图的数学公式可以写为如下形式：

$$\boldsymbol{I}=\alpha_p\boldsymbol{F}+(1-\alpha_p)\boldsymbol{B} \tag{7.1}$$

其中，$\boldsymbol{F}$ 和 $\boldsymbol{B}$ 分别对应于图像分割得到的前景和背景。

对于一般的彩色图像而言，每个像素有 R、G、B 三个通道，那么根据公式(7.1)就可以建立如下形式的抠图方程组：

图 7.1　图像抠图

$$\begin{cases} I_r = \alpha_p F_r + (1-\alpha_p) B_r \\ I_g = \alpha_p F_g + (1-\alpha_p) B_g \\ I_b = \alpha_p F_b + (1-\alpha_p) B_b \end{cases} \tag{7.2}$$

这里总共需要求解 7 个未知量，但该方程组只存在 3 个等式。因此，抠图一般被认为是一个高度“病态”的问题，从而严重影响了抠图值的正常求解。

事实上，抠图在很多影像处理中都有着重要的应用。这是由于很多的图像/视频画面中往往包含尺度上很微小的物体，例如毛发、火焰等。通过抠图操作，可以为这类微小物体提供更为精细的划分方式。这种微小特征通常是由运动模糊或者物体本身尺度比较细小所造成的。例如，头发引起的像素部分被遮挡，使得该像素也混杂了头发以外的成分。在这种情况下，一个像素就变成了多个超级采样亚像素的集合，而对这些亚像素进行分割所得到的数值，就决定了该像素的抠图值。图 7.2 展示了对同一幅输入图像分别进行分割和抠图操作，以及将前景进行合成之后的结果。可以看到，将由抠图操作得到的前景合成到另一幅新的图像时，毛发边缘更加自然柔和，更像是直接拍摄所得到的图像。

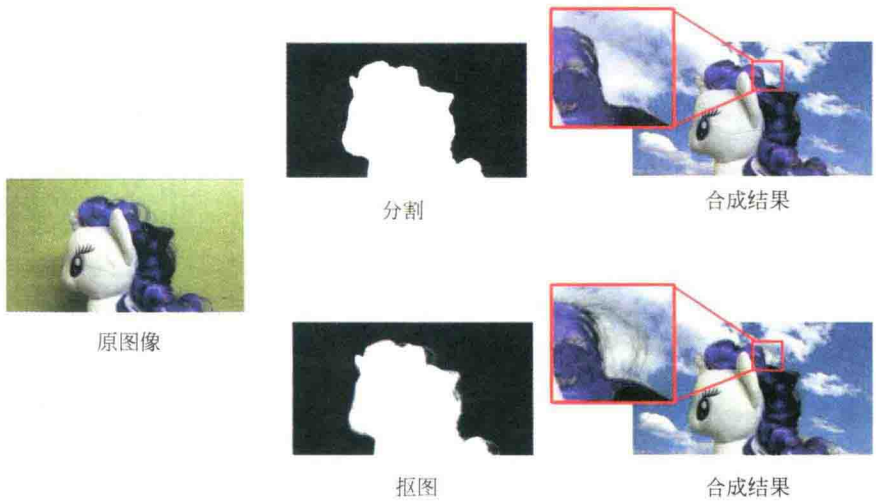


图 7.2　图像分割与抠图

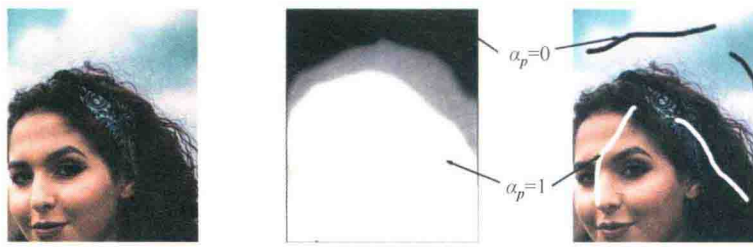
由于抠图本身是病态问题，在求解时的基本策略是引入一些先验知识作为约束。这样就可以将病态问题转化为数学上可求解的问题。具体来讲，通过增加先验知识的约束来减少未知量或者增加约束方程个数的方式，可以对式(7.1)所示的抠图函数进行优化，进而得到有效的抠图结果。常见的增加先验约束的方式大致可以分为两大类：蓝屏抠图

(这时背景 $\boldsymbol{B}$ 是已知的,前景 $\boldsymbol{F}$ 和抠图值 α_p 是未知的)和自然图像抠图(这时背景 $\boldsymbol{B}$、前景 $\boldsymbol{F}$ 和抠图值 α_p 都是未知的)。在这两类方法中,自然图像抠图相对更困难些。由于背景未知,自然抠图往往借助三色图(Trimap)或者笔画图(Stroke Map)提供的前景和背景内容作为先验知识,以此来辅助公式(7.2)中抠图值的求解。

如图 7.3 所示,基于三色图的方法是采用三种不同强度的灰度值分别标识输入图像中的前景、背景和过渡区域。其中,前景和背景对应于硬分割时具有明确抠图值($\alpha_p=1$ 和 $\alpha_p=0$)的区域;过渡区域的抠图值 $0\leqslant\alpha_p\leqslant1$ 则是未知的,需要借助已知的前景和背景区域作为先验约束进行计算。典型的基于三色图的方法包括贝叶斯抠图、泊松抠图等。

基于笔画图的方法则是通过交互的方式指定前景、背景和过渡区域。通常采用在图像上进行勾画的方式,选取笔画覆盖区域作为相应的种子区域,并以此建立先验约束条件。基于笔画的方法为抠图操作提供了交互上的更大灵活性。典型的方法包括最小二乘抠图、闭形式抠图等。此外,针对自然图像也有一些完全自动的抠图方法,例如闪光抠图等。

下面具体介绍这些抠图方法。



(a) 输入图像　　(b) 三色图　　(c) 笔画图

图 7.3　图像抠图中的三色图和笔画图

7.1.1　蓝屏抠图

蓝屏抠图技术最早在 20 世纪 50 年代由美国 MGM 公司的 Petros Vlaos 等人在电影制作过程中最先使用并逐步发展起来。这项技术也曾经荣获奥斯卡终身成就奖。事实上,蓝屏抠图是采用单一颜色作为背景拍摄的,从而能够方便地抠取前景。如图 7.4 所示,待拍摄的物体需要放置于单色背景前面,往往是一块蓝色或绿色的布,同时要求物体表面不包含背景颜色分量,这样就可以直接对前景物体进行抠图操作。

图 7.4　蓝屏抠图

根据公式(7.2),如果是在采用蓝屏背景的情况下,背景颜色 B 和前景部分颜色分量就会变为已知量。例如,使用蓝屏时背景为纯蓝色,那么背景颜色 B 的三个通道中的红

色和绿色通道分量就变为 0。因此，对应于每个像素的 3 个方程中的未知量就会由 7 个减少为 4 个。此外，如果要求前景物体不含蓝色成分，也就是前景的蓝色通道分量为 0，那么就得到了如下只包含 3 个未知量的线性方程组：

$$\begin{cases} \boldsymbol{I}_r = \alpha_p \boldsymbol{F}_r \\ \boldsymbol{I}_g = \alpha_p \boldsymbol{F}_g \\ \boldsymbol{I}_b = (1-\alpha_p) b \end{cases} \tag{7.3}$$

其中，b 是蓝色背景的蓝色分量。这样，通过公式(7.3)便可以依次计算 α_p、$\boldsymbol{F}_r$ 和 $\boldsymbol{F}_g$，从而得到物体的抠图结果。

例题 7-1　图像的蓝屏抠图。

问题：假设右图所示的图像 $\boldsymbol{I}$ 的背景为纯绿色，写出蓝屏抠图得到前景物体的伪代码。

解答：对于输入图像的每一个像素 $\boldsymbol{I}[i][j]$ 计算抠图值 $\alpha[i][j]$ 和前景物体像素 $\boldsymbol{F}[i][j]$ 颜色值的伪代码是：

```
function blueScreen (I, m, n)
    α ← 0
    for i ← 0 to m-1 do
        for j ← 0 to n-1 do
            α[i][j]  ← 1-I[i][j].g/255
            F[i][j].r ← I[i][j].r / α[i][j]
            F[i][j].b ← I[i][j].b / α[i][j]
            F[i][j].g ← I[i][j].g
        end for
    end for
end function
```

□

7.1.2　基于三色图的自然图像抠图

蓝屏抠图技术虽然方法简单，但使用条件却有严格的限制。这是因为必须采用纯色(蓝色或绿色)背景。然而对于一般的自然图像，背景往往比较复杂，因此很多场合无法采用蓝屏抠图技术。这时，可以通过三色图的方式引入一些适当的先验知识，进而增加公式(7.2)求解时的已知条件。在这种三色图提供的约束条件的基础上，再进一步采用贝叶斯优化、泊松优化等方法计算每个像素的抠图值。

1. 贝叶斯抠图

贝叶斯抠图是根据输入图像对应的三色图，借助贝叶斯推理来计算待求解区域 Ω 内的每个像素的抠图值。对于每个像素 $p \in \Omega$，在其最接近的前景区域 $\boldsymbol{F}$ 和背景区域 $\boldsymbol{B}$，分别寻找相应的像素颜色值作为初始值，然后借助贝叶斯定理计算每个像素的抠图值 α_p。然后，采用计算所得的抠图值更新三色图，按照求解出的抠图值修改三色图的像素灰度值。接着，基于新的三色图，再次利用贝叶斯定理求解抠图值。不断重复这个过程，直到三色图的灰度值不再发生变化，最终就可以得到相应的抠图结果。

在贝叶斯抠图过程中，每一次循环都需要借助贝叶斯公式计算抠图值。假设在第 n 次循环过程中，三色图前景区域为 $\boldsymbol{F}_n$，背景区域为 $\boldsymbol{B}_n$，那么可以通过高斯混合函数对前景和背景区域的颜色分布分别进行建模。以 R、G、B 任一通道为例，其强度值对应的高斯分布函数的方差和均值分别记为 $(\Sigma_{\boldsymbol{F}_n})^{-1}$、$(\Sigma_{\boldsymbol{B}_n})^{-1}$ 和 $\bar{\boldsymbol{F}}_n$、$\bar{\boldsymbol{B}}_n$。假设前景区域的像素数为 M，背景区域的像素数为 N，那么前景和背景区域的均值分别表示为 $\bar{\boldsymbol{F}}_n = \sum_{i=1}^{M} \boldsymbol{F}_n^i / M$ 和 $\bar{\boldsymbol{B}}_n = \sum_{i=1}^{N} \boldsymbol{B}_n^i / N$，对应的方差则是 $\Sigma_{\boldsymbol{F}_n} = \sum_{i=1}^{M} (\boldsymbol{F}_n^i - \bar{\boldsymbol{F}})(\boldsymbol{F}_n^i - \bar{\boldsymbol{F}})^{\mathrm{T}} / M$ 和 $\Sigma_{\boldsymbol{B}_n} = \sum_{i=1}^{N} (\boldsymbol{B}_n^i - \bar{\boldsymbol{B}})(\boldsymbol{B}_n^i - \bar{\boldsymbol{B}})^{\mathrm{T}} / M$。它们分别描述了前景区域和背景区域的颜色特征。那么，对于待求解区域的每个像素 $\boldsymbol{p}$，其颜色值与前景和背景的相似性可以通过如下公式计算：

$$\begin{cases} L_p^{\boldsymbol{F}_n} = -(\boldsymbol{I}_p - \bar{\boldsymbol{F}}_n)^{\mathrm{T}} (\Sigma_{\boldsymbol{F}_n})^{-1} (\boldsymbol{I}_p - \bar{\boldsymbol{F}}_n)/2 \\ L_p^{\boldsymbol{B}_n} = -(\boldsymbol{I}_p - \bar{\boldsymbol{B}}_n)^{\mathrm{T}} (\Sigma_{\boldsymbol{B}_n})^{-1} (\boldsymbol{I}_p - \bar{\boldsymbol{B}}_n)/2 \end{cases} \tag{7.4}$$

这个公式描述了由当前三色图定义的前景和背景颜色表示该像素颜色时的组成比例，也就是该像素分别属于前景和背景区域的概率。

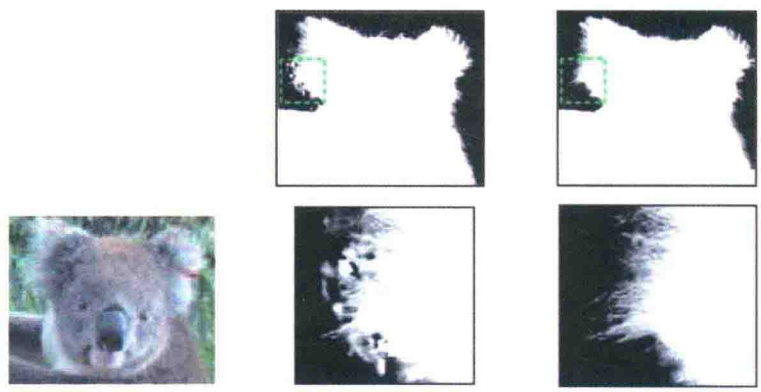
根据贝叶斯定理，在已知像素 $\boldsymbol{p}$ 的颜色值的前提下，前景 $\boldsymbol{F}$、背景 $\boldsymbol{B}$ 以及抠图值 α_p 满足如下公式：

$$P(\boldsymbol{F}_n, \boldsymbol{B}_n, \alpha_p \mid \boldsymbol{I}_p) = P(\boldsymbol{I}_p \mid \boldsymbol{F}_n, \boldsymbol{B}_n, \alpha_p) P(\boldsymbol{F}_n) P(\boldsymbol{B}_n) P(\alpha_p) / P(\boldsymbol{I}_p) \tag{7.5}$$

其中，P 表示像素颜色的后验概率分布。进一步，利用公式(7.4)和经典的最大后验概率公式，就可以将抠图值的计算转化为如下能量最大化的公式：

$$\underset{\boldsymbol{F}_n, \boldsymbol{B}_n, \alpha_p}{\operatorname{argmax}} L(\boldsymbol{I}_p \mid \boldsymbol{F}_n, \boldsymbol{B}_n, \alpha_p) + L(\boldsymbol{F}_n) + L(\boldsymbol{B}_n) + L(\alpha_p) \tag{7.6}$$

进一步计算在当前三色图约束条件下，已知前景和背景状态下相应的抠图值 α_p。根据得到的 α_p，更新三色图中前景和背景区域。通过不断迭代，最终得到抠图结果，如图 7.5(b)所示。



(a) 输入图像　(b) 贝叶斯抠图　(c) 泊松抠图

图 7.5　同一幅图像的贝叶斯抠图和泊松抠图结果(图片来自[68])

2. 泊松抠图

泊松抠图也是基于三色图提供的约束条件，但是将抠图问题转化为泊松方程形式进

行求解。这种方法假设前景区域和背景区域颜色的变化是均匀的，也就是说$\nabla \boldsymbol{F}$和$\nabla \boldsymbol{B}$都可以近似为常值。那么，对公式(7.1)进行求导并化简，可以得到如下表达式：

$$\nabla \alpha_p \approx \frac{1}{\boldsymbol{F}-\boldsymbol{B}} \nabla \boldsymbol{I}_p \tag{7.7}$$

其中，∇表示梯度算子，表明抠图值的梯度和图像颜色梯度之间具有简单的线性正比关系。

基于公式(7.7)，可以将待求解区域Ω内每个像素抠图值α_p的计算转化为泛函最小化问题，具体表示为

$$\underset{\alpha_p}{\operatorname{argmin}} \iint_{p \in \Omega} \left|\left| \nabla \alpha_p - \frac{1}{\boldsymbol{F}_p - \boldsymbol{B}_p} \nabla \boldsymbol{I}_p \right|\right|^2 \mathrm{d}p \tag{7.8}$$

其中，$\boldsymbol{F}_p$和$\boldsymbol{B}_p$是该像素的前景和背景颜色分量。数学上，公式(7.8)的求解可以转化为关于α_p的泊松方程，记为

$$\Delta \alpha_p = \operatorname{div}\left(\frac{\nabla \boldsymbol{I}_p}{\boldsymbol{F}_p - \boldsymbol{B}_p}\right) \tag{7.9}$$

其中，div 是散度算子。

为了求解上述泊松方程，仍然采用迭代优化方法，分别计算抠图值、前景和背景区域。具体来讲，这里根据三色图中已知的前景和背景区域找到距离最近像素的颜色值分别作为$\boldsymbol{F}_p$和$\boldsymbol{B}_p$的初始值，那么公式(7.9)的求解就变成了关于α_p的线性最小二乘优化问题。然后，通过共轭梯度法就可以方便地计算公式(7.9)对应的最优解，将其作为待求解区域像素的抠图值。最后，根据求解的抠图值及预设的关于前景像素和背景像素的抠图阈值，更新三色图中前景区域和背景区域，以此得到新的三色图。接下来与贝叶斯抠图类似，通过不断迭代该计算过程，最终得到抠图结果。图 7.5(c)展示了泊松抠图结果。相比于贝叶斯抠图，泊松抠图方法对于图像中微小特征的抠取更加准确，使得前景和背景区域的区分度更好。

7.1.3　基于笔画的自然图像抠图

虽然三色图能够为图像抠图提供足够的约束条件，但根据输入图像生成三色图并不是很容易，尤其是当抠取物体的轮廓形状比较复杂时。相比于三色图，笔画则提供了一种更为简洁的交互方式，使得用户能够通过几笔简单的勾画，指定前景和背景的种子区域。这类方法拿笔画覆盖区域定义一些先验条件，然后通过求解抠图方程计算其他区域像素的抠图值。常用的基于笔画的自然图像抠图方法有最小二乘抠图、闭形式抠图等。下面分别介绍这两种方法。

1. 最小二乘抠图

最小二乘抠图是将抠图方程转化为以抠图值α_p作为变量的二次函数。该函数忽略了未知区域像素的前景和背景颜色值对于α_p的影响。具体来讲，这种方法利用笔画覆盖区域的前景和背景颜色作为约束条件，计算有效的抠图结果。它假设相邻像素具有连续的α_p数值，而且由前景和背景区域提供的颜色具有与待求解像素颜色最小的差异。为此，如图 7.6(a)所示，通常对于每一个像素$\boldsymbol{p}$，从背景和前景区域分别挑选N_p个距离最近的像素，利用它们提供的颜色作为先验信息。这里挑选的前景和背景区域像素集合分

别记为$\{\boldsymbol{F}_p^i\}$和$\{\boldsymbol{B}_p^i\}$。那么，可以建立如下形式的抠图函数：

$$E=\sum_{p\in\Omega}\Big(\sum_{i=1}^{N_p}\|\boldsymbol{I}_p-\hat{\boldsymbol{I}}_p^i\|^2/\sigma_p^2+\lambda\sum_{q\in N(p)}(\alpha_p-\alpha_q)^2/\|\boldsymbol{I}_p-\boldsymbol{I}_q\|\Big) \tag{7.10}$$

其中，$\hat{\boldsymbol{I}}_p^i=\alpha_p\boldsymbol{F}_p^i+(1-\alpha_p)\boldsymbol{B}_p^i$是由已知的前景和背景先验所得到的图像；$\sigma_p$是在邻域范围内颜色分布的标准方差；$N(\boldsymbol{p})$是像素$\boldsymbol{p}$的邻域。

因此，对于未知像素，利用前景笔画和背景笔画覆盖的像素颜色值作为已知条件，式(7.10)就会变成关于未知量α_p的二次函数。借助最小二乘优化，就可以计算未知区域像素的抠图值α_p。这种方法也可以采用迭代的方式，反复更新前景区域和背景区域，不断地对未知区域的抠图值进行优化求解，得到最终的抠图结果(如图 7.6(b)所示)。

(a) 前景和背景约束　　(b) 抠图结果

图 7.6　最小二乘抠图(图片来自[69])

2. 闭形式抠图

闭形式抠图采用简化的前景和背景颜色模型，将抠图方程转化为具有显式表达式的形式。如果输入图像是灰度图，该方法则在局部窗口范围内，假设前景和背景颜色值近似为常数，也就是说局部窗口内的图像是单色图。那么，对于局部窗口内的每个像素，相应的抠图值可以近似地写成关于像素灰度值的线性函数，记为$\boldsymbol{I}_p\approx\alpha_p\boldsymbol{F}+(1-\alpha_p)\boldsymbol{B}$，其中$p\in w$是局部窗口内的像素。进一步，可以得到每个像素对应的抠图值，记为$\alpha_p=a\boldsymbol{I}_p+b$。这里的两个系数分别是$a=1/(\boldsymbol{F}-\boldsymbol{B})$和$b=-\boldsymbol{B}/(\boldsymbol{F}-\boldsymbol{B})$。在此基础上，对于未知区域的像素可以建立如下抠图方程：

$$E(\alpha,a,b)=\sum_i\Big[\sum_{p\in w_i}(\alpha_p-a_i\boldsymbol{I}_p-b_i)^2+\varepsilon a_j^2\Big] \tag{7.11}$$

式中，w_i表示第i个局部窗口，一般设置为3×3的窗口；第二项是正则项，ε是权因子。式(7.11)是线性二次函数，通过使该函数取值最小化来计算所有像素对应的抠图值，记为$\{\alpha_p\}=\mathrm{argmin}\boldsymbol{\alpha}^{\mathrm{T}}\cdot\boldsymbol{L}\cdot\boldsymbol{\alpha}$。这里$\boldsymbol{\alpha}$是由所有像素抠图值组成的列向量；$\boldsymbol{L}$是由像素灰度值定义的方阵。

上述思想推广到一般的彩色图像，对于局部窗口像素的抠图值可以采用R、G、B三个通道的线性函数近似。如图 7.7 所示，对于每个像素$\boldsymbol{p}$记为$\alpha_p=a^R\boldsymbol{I}_p^R+a^G\boldsymbol{I}_p^G+a^B\boldsymbol{I}_p^B+b$。根据式(7.11)，就可以得到彩色图像 R、G、B 三个通道对应的抠图方程：

$$E(\alpha,a,b)=\sum_{i\in I}\Big(\sum_{p\in w_i}(\alpha_p-a_i^R\boldsymbol{I}_p^R-a_i^G\boldsymbol{I}_p^G-a_i^B\boldsymbol{I}_p^B-b_i)^2+\varepsilon a_i^{R2}+\varepsilon a_i^{G2}+\varepsilon a_i^{B2}\Big) \tag{7.12}$$

事实上，公式(7.12)是以抠图值作为变量的二次函数。这里是将用户交互笔画作为

已知的前景和背景区域，那么其对应的抠图值分别是 0 或 1，并以此作为函数优化时的约束条件。此外，由于公式(7.12)是关于抠图值的线性二次函数，可以采用共轭梯度法得到未知区域所有像素抠图值的显式解，以及相应的前景区域和背景区域。

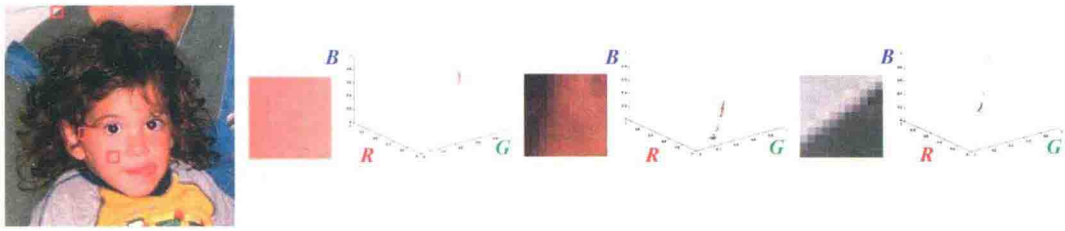


图 7.7　局部窗口的颜色的线性分布(图片来自[70])

7.1.4　基于闪光的自然图像抠图

前面介绍的基于三色图和笔画的自然图像抠图，都还需要以交互的方式提供先验知识作为约束条件。如果完全自动地进行自然图像抠图，则往往需要同时提供多幅图像，其中最典型的方法是闪光抠图。这种方法要对场景同时拍摄正常和闪光两幅图像，进而引入额外的约束条件。具体来讲，对于同一个需要抠取的物体，在开启和关闭闪光的情况下，分别拍摄一幅图像，记为闪光图像 $\boldsymbol{I}^f$ 和非闪光图像 $\boldsymbol{I}$(如图 7.8(a)和(b)所示)。然后，将两幅图像的差别作为抠图方程中的约束条件。

根据公式(7.1)，对应于闪光图像的抠图方程是 $\boldsymbol{I}^f=\alpha\boldsymbol{F}^f+(1-\alpha)\boldsymbol{B}^f$，而非闪光图像的抠图方程是 $\boldsymbol{I}=\alpha\boldsymbol{F}+(1-\alpha)\boldsymbol{B}$。此外，由闪光图像减去非闪光图像，就可以得到纯闪光图像 $\boldsymbol{I}'=\boldsymbol{I}^f-\boldsymbol{I}$，如图 7.8(c)所示。该图像记录了使用闪光时的补光量，因而背景是纯黑色，由此可以得到 $\boldsymbol{B}^f=\boldsymbol{B}$。这也是闪光抠图方法中最重要的约束条件。然后在计算抠图值时，仍然可以采用贝叶斯抠图的优化方法。不同之处在于，在公式(7.5)的基础上，还需要额外增加纯闪光图像引入的后验概率，具体表示为 $L(\boldsymbol{I}'\mid\alpha,\boldsymbol{F}')=-\|\boldsymbol{I}'-\alpha\boldsymbol{F}'\|/\sigma_f^2$。最后通过贝叶斯优化，得到最终的抠图结果(如图 7.8(d)所示)。事实上，这种方法是通过闪光来增加抠图时的约束条件，从而能够实现自动抠图。

(a) 闪光图像　(b) 非闪光图像　(c) 纯闪光图像　(d) 抠图结果

图 7.8　自动闪光抠图(图片来自[71])

7.1.5　视频抠图

图像抠图方法推广到视频，就可以对视频帧画面中的物体进行抠图处理。视频抠图的关键在于如何保持帧间抠图值的一致性，也就是说，抠图值要随着帧间运动做相应的连

续变化，这样才能确保抠取时前景和背景的一致性。视频抠图的基本思想是首先在关键帧上获取相应的抠图值，然后结合视频运动在帧间传播抠图值，从而形成时空连续的抠图值。

如图 7.9 所示，将抠图值从关键帧传播到其他帧时，往往是以关键帧的抠图值作为初始值，通过双向传播插值中间帧的抠图值。在此基础上，进一步精细处理物体边界处的抠图值，生成中间帧的抠图结果。不同的视频抠图方法往往会借助不同的运动形式进行帧间传播，例如采用光流方法、视频体方法等。

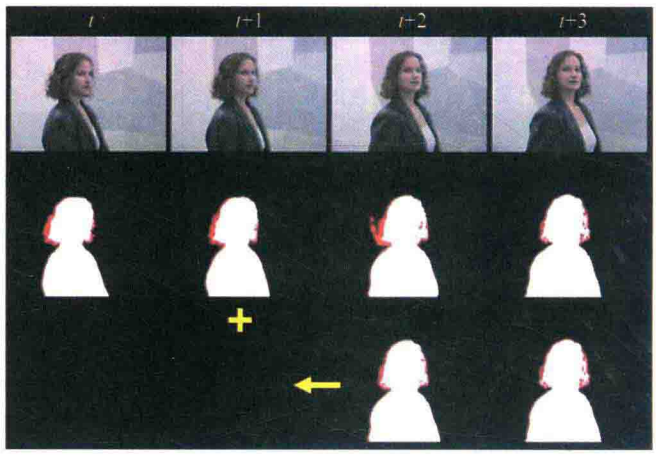


图 7.9 基于光流的视频抠图

7.2 影像缩放

图像/视频在不同终端显示时，由于设备制造的差异，产生的画面尺寸会存在不一致的问题(如图 7.10(a)所示)。这样在不同终端显示时，需要调整图像/视频的画面尺寸。影像缩放就是通过减少或者扩展原始图像的边界大小，或者增删原始画面的内容，使缩放后的影像能够适应于不同的显示屏幕，从而满足用户在不同终端观看影像内容的要求。

(a) 不同尺寸的显示终端　　(b) 不同缩放结果

图 7.10 图像缩放

传统图像处理的方法在进行影像缩放时，大多是基于尺寸裁剪的方式，也就是对输入的图像或者视频帧画面进行等比例缩放，或者直接将图像按照显示屏的尺寸进行裁剪。这些处理方式不可避免地造成缩放后图像内容的严重损失(如图 7.10(b)所示)。基于图形学的方法则会根据影像中画面内容的差异，在不影响缩放后观看效果的前提下，对图像或视频帧画面进行增删或变形，以满足指定的缩放尺寸要求。下面介绍几种典型的影像缩放方法。

7.2.1 图像缝隙增删

缝隙增删的基本思想是在图像的水平和竖直方向，不断插入或删除缝隙来扩大或减小图像。这里所谓的缝隙(Seam)，是指图像中路径连通的低能量像素通路，同时该通路使得每行或者每列只包含一个像素。因此，如图 7.11 所示，水平方向的缝隙增删会扩大或缩小图像的宽度，而竖直方向的缝隙增删则会扩大或缩小图像高度。那么，综合使用水平或竖直方向的缝隙增删，就能够实现图像在两个方向上的尺寸缩放。

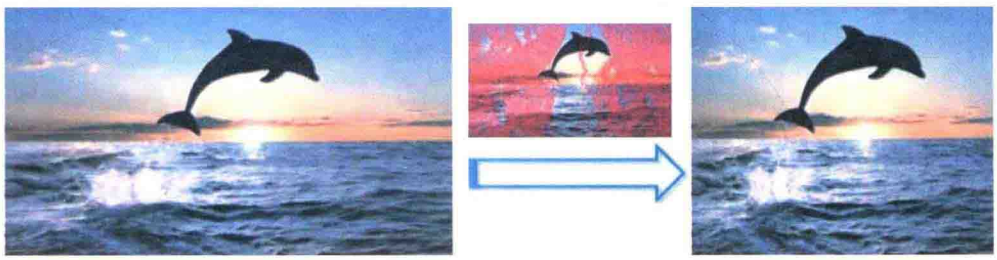


图 7.11 删除缝隙减小图像宽度(图片来自[73])

显而易见，缝隙增删方法的关键是如何定义缝隙，使得缩放后的图像内容具有自然的视觉效果。典型的缝隙定义，是指在图像中穿越较低视觉显著性区域的那些连线。这里的显著性是指图像中容易引起视觉注意力的机制，通常定义为图像梯度的大小，或者其他基于视觉感知机理的模型。具体来讲，图像 $\boldsymbol{I}$ 在像素位置 $\boldsymbol{p}=(x,y)$ 处的视觉显著性可以通过梯度计算来表示为 $\boldsymbol{e}(\boldsymbol{I}_p)=|\partial \boldsymbol{I}_p/\partial x|+|\partial \boldsymbol{I}_p/\partial y|$。

在此基础上，需要在水平或者竖直方向寻找连续的缝隙。以竖直方向为例，为了保持缝隙的连续性，往往将上下相邻的一组像素集合作为候选的缝隙。因此，给定一条缝隙，它的视觉显著性可以定义成

$$E(\boldsymbol{p}=(x(i),i))=\sum \boldsymbol{e}(\boldsymbol{I}_p),\quad 1\leqslant i\leqslant H \tag{7.13}$$

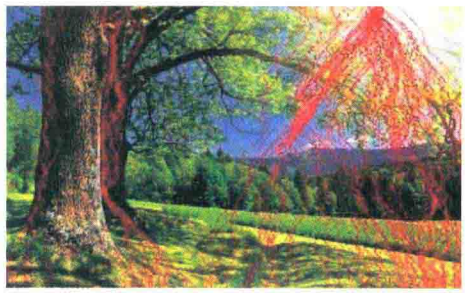
其中，H 是图像高度，横坐标 $x(i)$ 约束了缝隙在竖直方向必须是连续不间断的。这样就保证了缝隙在竖直方向是路径连通的通路。那么，最优的缝隙就是使得公式(7.13)取值最小的像素所组成的集合。为了优化该公式，可以通过动态规划的方法进行快速计算，然后依次将找到的缝隙在图像中做插入或删除操作，由此得到宽度扩大或缩小的图像。因为这些缝隙只是经过一些视觉不显著的区域，多数情况下不会引起强烈的视觉反应，所以通常不会影响图像中重要内容在缩放后的观看。

然而，基于缝隙增删的图像缩放方法效率较低。这主要是由于每次只能增加或删除

一条缝隙，是一个顺序执行的过程。此外，对于一些显著的物体结构属性，如轮廓的直线性，由这种方法得到的缩放结果也缺乏较好的直线保持特性，以致原始的直线出现错位或断裂等问题。

例题 7-2　基于缝隙增删的图像缩放。

问题：如右图所示的输入图像，写出采用动态规划计算竖直方向最优缝隙（红线所示）并进行宽度缩小的伪代码。



解答：假设输入图像 I 的宽和高分别是 m 和 n，那么在竖直方向计算 k 条最优缝隙，并进行删除以缩小宽度的伪代码是

```
function optimalSeam (I)
    saliency ← gradient (I)                          /* 通过梯度计算每个像素的显著性 */
    for i ← 0 to m-1 do                              /* 动态规划记录路径显著性 */
        for j ← 0 to n-1 do
            if j ==0 then
                dyn[j][i] ← saliency[j][i];
                else if i ==0 then
                    if dyn[j-1][i] <=dyn[j-1][i +1] then
                        dyn[j][i] ← dyn[j-1][i] +saliency[j][i]
                    else
                        dyn[j][i] ← dyn[j-1][i +1] +saliency[j][i]
                    end if
                else if i ==m-1 then
                    if dyn[j-1][i] <=dyn[j-1][i-1] then
                        dyn[j][i] ← dyn[j-1][i] +saliency[j][i]
                    else
                        dyn[j][i] ← dyn[j-1][i-1] +saliency[j][i]
                    end if
                else if dyn[j-1][i -1] <=dyn[j-1][i] and dyn[j-1][i -1] <=
                                dyn[j-1][i +1]) then
                    dyn[j][i] ← dyn[j-1][i-1] +saliency[j][i]
                else if dyn[j-1][i +1] <=dyn[i-1][i] and dyn[j-1][i +1] <=
                                dyn[j-1][i-1]) then
                    dyn[j][i] ← dyn[j-1][i +1] +saliency[j][i]
                else
                    dyn[j][i] ← dyn[j-1][i] +saliency[j][i]
                end if
            end if
        end for
    end for
    for l ← 1 to k do
        s ← findSeam (I, dyn)                        /* 寻找当前最优的路径作为缝隙 */
        I ← I-s                                      /* 删除该缝隙得到宽度缩小的图像 */
```

```
    end for
end function
```

□

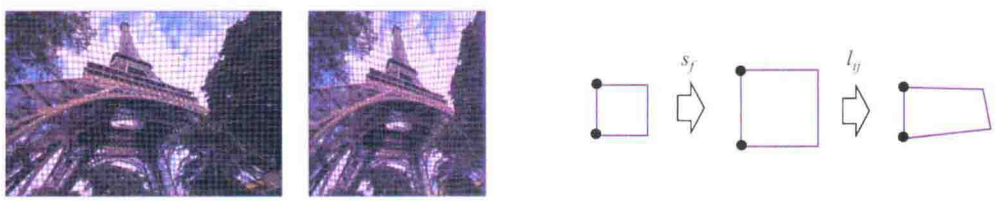
7.2.2　图像网格变形

图像网格变形是以缩放后的目标尺寸为约束，对原始图像进行变形操作。这就把尺寸改变的过程转化为图像内容中形状的变化。这种变形方法一方面使得缩放后的图像满足指定的尺寸要求，另一方面也要确保原始图像中物体的形状不发生较大的变化，也就是尽量地保持物体的结构不变。常用的网格变形方法是采用图像嵌入的四边形网格变形来驱动图像的尺寸缩放。

图形学中的变形方法大多基于网格驱动的策略。如图 7.12 所示，通过对网格顶点、边等基本几何元素的空间位置的改变，实现网格形状的变化。因此，图像变形时首先将图像嵌入一个预设的网格。然后为了实现形状保持的变形，需要采用相似变换对网格顶点、边等的位置做变化，进而驱动整个图像的变形。具体来讲，网格顶点坐标$\{\boldsymbol{v}_i\}$变换后的坐标$\{\boldsymbol{v}_i'\}$应当是满足如下变形函数的最优解：

$$\operatorname{argmin} \sum_{\{i,j\}\in E} \| (\boldsymbol{v}_i' - \boldsymbol{v}_j') - s_j(\boldsymbol{v}_i - \boldsymbol{v}_j) \|^2 + \sum_{\{i,j\}\in E} \| (\boldsymbol{v}_i' - \boldsymbol{v}_j') - l_{ij}(\boldsymbol{v}_i - \boldsymbol{v}_j) \|^2 \tag{7.14}$$

其中，E 是所有网格边$<\boldsymbol{v}_i, \boldsymbol{v}_j>$的集合；$s_j$ 是按等比例缩放达到目标尺寸的最小比例因子；l_{ij} 是缩放后边长变化的倍数。



(a) 网格变形　　(b) 形状保持的格子变换

图 7.12　网格变形驱动图像缩放

公式(7.14)中的第一项约束每个四边形的边在变形时只会发生伸缩和平移，这就意味着每一个四边形格子只是根据比例因子 s_j 做等比例缩放。由于边长比例不变，所以这样能够保持图像内容不发生明显的形变。第二项是在图像发生不等比例缩放操作时，网格的每一个四边形的边要按照指定的缩放倍数 l_{ij} 进行变化，也就是能够满足目标尺寸的倍数。显然，如果只有这两个能量项，变换函数的最优解是平凡的零解。为了获得有效的非平凡解，需要进一步在公式(7.14)中添加额外的约束条件。这里使用图像缩放后的边界位置作为约束，具体包括缩放后图像左上角顶点的位置 $\boldsymbol{v}'_0 = (0,0)^{\mathrm{T}}$ 和右下角顶点的位置 $\boldsymbol{v}_{\mathrm{end}}' = (N', M')^{\mathrm{T}}$，以及水平和竖直边界上每个网格顶点的位置。这些顶点位置一方面作为已知条件使得公式(7.14)具有非平凡解，另一方面也限定了缩放后图像的尺寸。在此基础上，可以计算满足指定边界条件的网格顶点位置，进而驱动图像变形。

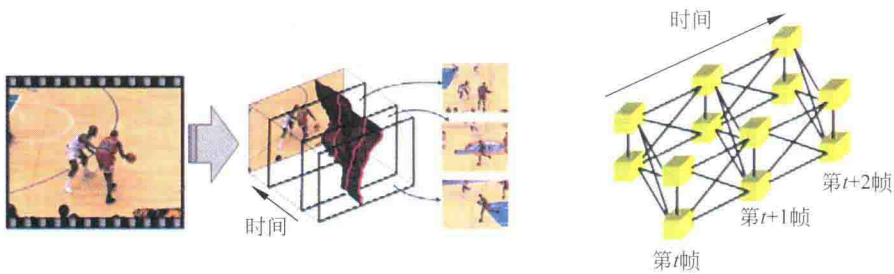
为了在图像变形过程中使显著结构发生尽可能小的变化，可以采用图像梯度和图像显著度相结合的度量来得到结构图。这样，在图像变形过程中，显著性较强的网格顶点位置发生的变动尽可能小。通过这种方式可以进一步提高图像缩放后的视觉效果，尤其是尺寸改变较大时对原始内容的观感。

由于带约束的变形函数是非线性的，因此需要指定一个初值。这里可以采用简单的等比例缩放后的网格顶点位置作为初始值，然后通过迭代优化，求得最终缩放后的图像。但是，这种方法的缺点也很明显。如图 7.13 所示，由于在变形过程中采用相似变换，图像中的一些直线结构可能被破坏。

图 7.13　基于网格变形的图像缩放产生直线扭曲

7.2.3　视频缩放方法

与图像缩放相比，视频缩放的主要难点在于逐帧使用前面介绍的图像缩放方法时缺乏帧间时序的一致性，导致缩放后的视频帧画面在播放时跳跃明显。因此，视频缩放的一个基本思路是将所有帧的集合看作一个视频体，然后再将图像缩放方法推广到视频缩放。这样就能够在一定程度上提高缩放操作时的时空一致性。例如，基于视频体的缝隙增删方法，也就是要在时空域选择一致的缝隙增删策略，从而达到时空一致的视频缩放（如图 7.14(a)所示）。



(a) 视频体中的缝隙　　(b) 三维图割模型

图 7.14　基于缝隙增删的视频缩放（图片来自[75]）

与图像采用动态规划求解不同，视频的缝隙通过图割优化来计算。视频体的最优缝隙需要满足两个条件：单调性，即每行只能有一个像素；连通性，即每条缝隙将图像划分为两个连通区域。首先以单帧图像为例，构造满足缝隙特点的图割模型，然后再将该模型推广到视频帧序列。

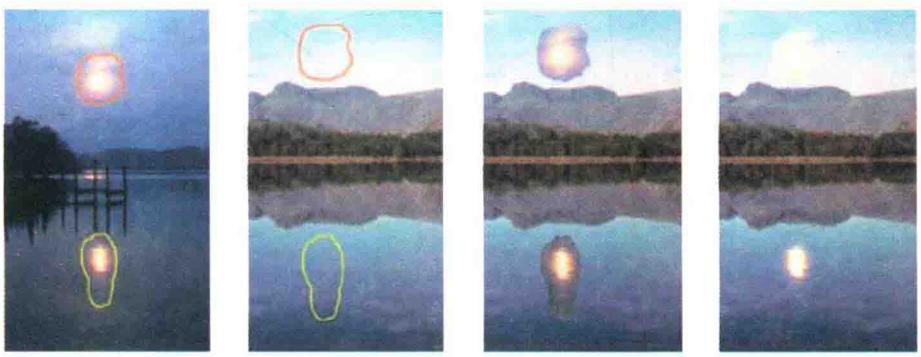
在经典的图割优化模型中，相邻像素之间通过带有权值的双向边进行连接，这样优化得到的图割边界不一定满足单调性和连通性。因此，为了满足上述两个特点，需要对其进行改造，将双向边中的一个方向的边的权值设为无穷大，而且将八邻域的连接边权值也设为无穷大。这样就可以满足相应的缝隙特点。进一步采用这种方式构造的图，经过图割优化得到的边界线就是由动态规划得到的缝隙。

为了处理连续的视频帧，需要将上述图的构造方式推广到三维视频体，以构造视频体对应的三维图。如图 7.14(b)所示，除了单帧图像相邻像素的边，还增加了相邻帧之间相邻像素之间的连接边，并设置相应的权值。这样，原图和缩放后的目标图分别通过无限加权的弧被创建并连接到图像的最左和最右列的像素里，形成三维体空间的图。同样利用图割优化，计算视频体在三维空间中的截面。该截面是贯穿起始和终止帧的面。它与中间每一帧都会形成一条交线，作为相应的缝隙。借助三维体空间图构造时的帧间连接方式，可以得到帧间连续变化的缝隙，从而有效避免了缩放后视频帧画面的跳跃。

7.3　影像融合

在影像创作过程中，往往需要将来自不同图像/视频的内容融合到另外指定的图像/视频中，从而生成新的图像/视频。在融合时，首先选取源图像中的区域作为源区域(如图 7.15(a)所示)，然后把它放入目标图像中(如图 7.15(b)所示)，再通过融合生成新的图像(如图 7.15(c)和图 7.15(d)所示)。

由于是对来自不同图像/视频的内容进行融合，需要使新生成的图像尽可能自然。而这个问题的关键之处是如何在源区域和目标图像重叠区域的边界处产生连续的过渡，这样才能使融合后的图像/视频看上去是自然的。直接将源区域置于目标图像中，就会产生如图 7.15(c)所示的不连续的边界过渡，具有很不自然的视觉效果。为了消除这种现象，传统图像处理的方法大多通过简单的透明度混合。例如，将源区域和目标图像的透明度分别设置为 0.5，然后在重合区域按照该透明度进行混合。虽然这种方式能够缓解边界处融合时的突变，但是没有考虑图像/视频本身内容的差异，融合结果仍然存在视觉的不连续性。另外一种是自适应融合，根据图像内容在边界处形成连续过渡，包括泊松图像融合、基于均值坐标插值的图形融合等。下面具体介绍这几种方法。



(a) 源区域　(b) 目标图像　(c) 简单融合　(d) 泊松融合

图 7.15　图像融合

7.3.1 泊松图像融合

泊松融合方法是借助梯度表示图像内容中的结构。如图 7.16 所示，它将源区域的梯度嵌入到目标图像，然后根据目标图像的颜色恢复源区域中的图像内容。这样既能够保持源区域的内容，又兼具目标图像的颜色特点，使得融合后的图像更加自然。这种图像融合方法的问题求解最终转化为计算泊松方程，因此称为泊松融合。

具体来讲，源区域的梯度记为 $\boldsymbol{v}_{(x,y)}=\nabla_{(x,y)}\boldsymbol{I}=(\partial\boldsymbol{I}/\partial x,\partial\boldsymbol{I}/\partial y)$，目标图像记为 $\boldsymbol{g}$，二者的重叠区域记为 Ω。那么在重叠区域，需要采用源区域的梯度代替目标图像在重叠区域的梯度，而融合后图像像素 $\boldsymbol{f}_{(x,y)}$ 的颜色值，则可以通过优化如下方程进行求解：

$$\underset{\boldsymbol{f}}{\operatorname{argmin}}\iint_{(x,y)\in\Omega}\left|\nabla\boldsymbol{f}_{(x,y)}-\nabla\boldsymbol{I}_{(x,y)}\right|^2\mathrm{d}x\mathrm{d}y,\quad \boldsymbol{f}\big|_{\partial\Omega}=\boldsymbol{g}\big|_{\partial\Omega} \tag{7.15}$$

这样最终融合后的图像按照源区域的梯度传递重叠区域边界处目标图像的颜色，达到自然的融合效果。数学上，公式(7.15)可以转化为泊松方程，表示为 $\Delta\boldsymbol{f}_{(x,y)}=\mathrm{div}\boldsymbol{I}_{(x,y)}$，其中 Δ 和 div 分别是定义在图像上的 Laplacian 算子和散度算子。在具体计算时，采用四邻域的离散差分运算计算相应的算子，最终求得融合后的图像。相比于简单融合，泊松融合结果既保持了原图像物体的内容结构，又符合目标图像的颜色，融合效果更加自然(如图 7.16 所示)。

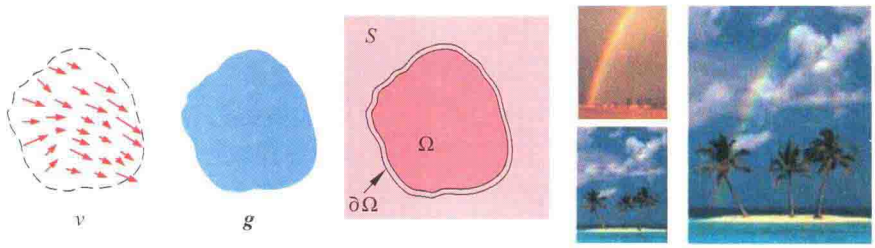


图 7.16 泊松图像融合(图片来自[76])

7.3.2 基于均值坐标插值的图像融合

泊松图像融合的计算过程实质上是通过求解给定边界值的泊松方程来实现。简单地讲，这一过程可以看作按照梯度分布对重叠区域的边界颜色进行插值。基于这种思路，图像融合可以转化为给定边界的插值问题，最后通过对颜色的插值生成融合后的图像。

几何插值方法中代表性的方法是基于多边形的均值坐标插值。给定一个多边形，假设顶点集合是 $\{\boldsymbol{c}_0,\boldsymbol{c}_1,\cdots,\boldsymbol{c}_{m-1}\}$，那么内部各点可以由多边形顶点的线性组合表示。具体来讲，如图 7.17 所示，多边形内部的点 $\boldsymbol{x}$ 可以表示为如下的线性组合形式：

$$\boldsymbol{x}=\sum_{i=0}^{m-1}\frac{w_i(\boldsymbol{x})}{\sum_{j=0}^{m-1}w_j(\boldsymbol{x})}\boldsymbol{c}_i \tag{7.16}$$

其中，组合系数 $w_i(\boldsymbol{x})$ 就是均值坐标，具体计算公式如下：

$$w_i(\boldsymbol{x})=\frac{\tan(\alpha_{i-1}/2)+\tan(\alpha_i/2)}{\|\boldsymbol{c}_i-\boldsymbol{x}\|} \tag{7.17}$$

这里 α_i 是该点与多边形顶点连线的夹角。该均值坐标可以看作三角形重心坐标的直接推广。它提供了线性表示多边形内部区域的方式。

在图像融合时，将源区域置于目标图像之上，重叠区域的边界就形成以像素为单位的多边形。沿着该多边形，计算源区域和目标图像对应像素的颜色差异。融合后的重叠区域内部则可以通过公式(7.16)对多边形顶点插值进行计算。为了加快插值计算过程，可以通过对重叠区域进行 Delaunay 三角化形成三角剖分(如图 7.17 所示)。这样就获得了重叠区域的离散图形表示。然后，对三角形顶点进行均值坐标插值，计算相应的融合颜色值。最后，对位于三角形内部的每个像素，则通过重心坐标插值计算融合后的颜色。这种方式大大加快了图像融合的计算效率，能够实现实时的融合效果预览。

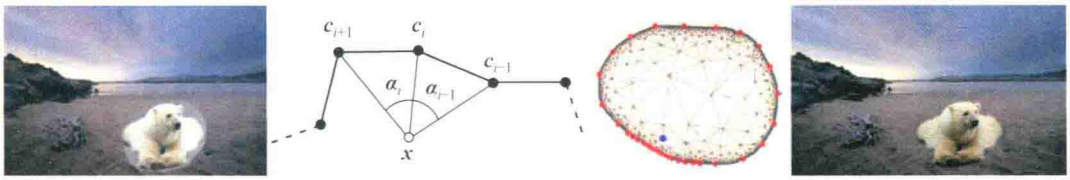


图 7.17　基于均值坐标插值的图像融合(图片来自[77])

例题 7-3　基于均值坐标插值的图像融合。

问题：如下图所示，写出采用均值坐标进行逐像素插值和图像融合的伪代码。

解答：假设源区域为 P_s，在目标图像上的重叠区域为 P_t，那么通过均值坐标插值得到融合图像 I 的伪代码如下。

```
function coordinateBlend (Ps, Pt, I)
    for each pixel p in Ps do
        λ0(p), ···, λm-1(p) ← MVC(p, ∂Ps)
    end for
    for each new Pt do
        for each vertex vi in ∂Pt do
            di ← f(vi) - g(vi)
        end for
        for each pixel p in Pt do
            r(p) ← Σ_{i=0}^{m-1} λi(p) · di
            I(p) ← g(p) + r(p)
```

```
        end for
    end for
end function
```

□

7.3.3 视频融合方法

与图像融合类似，视频融合是将源视频的区域放入目标视频中，从而生成包含两个视频内容的新视频。相比于图像融合，视频融合变得更加困难。这里的难点不仅在于融合后需要保持源区域的结构，而且也要和目标视频背景形成帧间连续的边界过渡，也就是融合操作的时空一致性问题。这样才能使得融合后的视频具有连续的帧间变化，用户观看时不会产生严重的帧间跳跃。此外，视频融合时还受到运动模糊、阴影等问题的影响，大大增加了视频融合的难度。接下来简要介绍一种基于梯度域的视频融合方法。

梯度域视频融合是图像梯度域融合的直接推广。它的基本思想也是在梯度域上混合源区域和目标视频背景，以保持源区域的结构。同时，利用光流在帧间重叠区域连续地传播边界的变化，使得融合后的结果在相邻帧间呈现自然的过渡。如图 7.18 所示，为了解决重叠区域边界的连续性，采用光流在帧间传播关键帧的重叠区域边界，并根据源区域和目标视频的颜色差异进行相应连续性优化处理。这样就能够确保融合边界的连续性。

对于重叠区域内梯度的连续性，采用低通滤波的方法对相邻的前后帧，以及原视频和目标视频帧的梯度进行加权混合，从而既保持了原始和目标视频内容的连续性，又兼具帧间变化的连续性。最后，采用前面介绍的均值坐标方法求解泊松方程，提高融合计算的效率。这样就获得了融合后的视频。

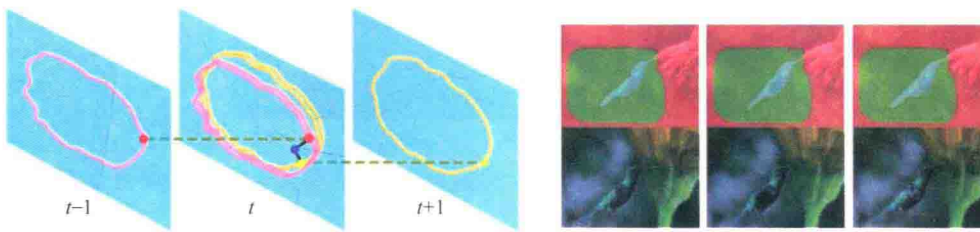


图 7.18 视频融合(图片来自[78])

7.4 影像拼接

普通相机拍摄照片的视角范围一般是 50°×35°，而人眼看到的视角范围则是 200°×135°。如图 7.19 所示，影像拼接就是将两个或者更多具有重合区域的图像/视频组合在一起，生成具有更大视角范围的图像/视频。因此，通过拼接就可以获得更大视角范围的图像/视频，甚至于 360°×180°的全景图像/视频。当然，这里进行拼接的图像/视频需要彼此具有明显的相同内容。

目前的影像拼接主要有硬件和软件两种方案。硬件方案是采用特殊的镜头，直接拍

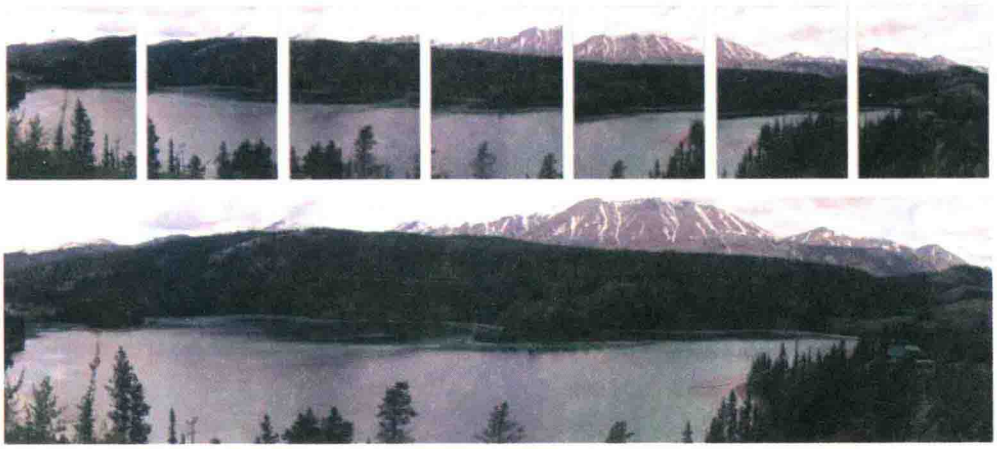


图 7.19　图像拼接

摄具有大范围视场角的图像或视频。例如，日本 Nikon 公司的鱼眼相机，可以拍摄 220°视场范围的照片。美国 Facebook 公司则推出了 360°全景相机，通过环绕一圈的镜头朝各个方向同时拍摄，直接获得全景照片。

软件方案是将多个相机拍摄的图像或视频通过变换组合在一起，形成视觉上更加完整、视角范围更大的图像或视频。由于成本较低、操作灵活，软件方案的使用更加广泛，特别适合于移动设备。基于软件方案的图像/视频拼接算法一般包含 4 个步骤：特征点检测、特征点匹配、对齐变换计算、图像/视频变换映射到共同区域。通过这 4 个步骤，最终就生成了拼接结果。其中，影响最后拼接效果的重要因素是对齐变换的选择。不同的拼接方法会采用不同的几何变换，常见的是单应变换（Homography）。单应变换具有 8 个自由度，描述了三维空间中的平面在不同视角之间的变换。

假设图像 $\boldsymbol{I}_2$ 与图像 $\boldsymbol{I}_1$ 之间根据特征点的对应关系进行单应变换 $\boldsymbol{H}_{21}$，那么拼接后的图像记为 $\boldsymbol{I}_1 \cup \boldsymbol{H}_{21} \cdot \boldsymbol{I}_2$，表示了两幅图像内容的组合。在图像拼接时，采用尽可能单应变换、形状保持的半单应变换、自适应混合变换等方法。这些方法大多是将图像置于网格之中，通过对网格顶点的变换驱动图像变换，进而实现两幅图像的拼接。视频拼接方法往往是在图像拼接的基础上，引入帧间连续性约束，得到时空一致的拼接效果。接下来具体介绍各种拼接方法。

7.4.1　尽可能单应变换的图像拼接

单应变换描述同一平面在不同视角图像之间的映射关系。因为在图像拼接时，如果对整幅图像只使用一个整体的单应变换，会影响重合区域特征点的对齐精度，同时在非重合区域也会产生比较大的扭曲。更为有效的图像拼接方法是采用局部单应变换集合取代单一的整体单应变换，对不同的区域施加尽可能满足单应变换性质的变换，以此来提高匹配特征点对齐时的准确性，减少形状扭曲。这种方法称为尽可能单应变换（As-projective-as-possible，APAP）。

给定对应特征点集合 $\{\boldsymbol{x}_i \leftrightarrow \tilde{\boldsymbol{x}}_i\}_{i=1}^{N}$，传统单应变换的计算是利用直接线性变换（Direct

linear transform，DLT）技术进行求解。该技术将单应变换矩阵 $\boldsymbol{H}$ 的 9 个元素看作向量 $\boldsymbol{h}=(h_1,h_2,\cdots,h_9)$，通过寻找满足代数误差函数 $\underset{\boldsymbol{h}}{\operatorname{argmin}}\sum_{i=1}^{N}\|\boldsymbol{a}_i\cdot\boldsymbol{h}\|^2$ 最小化的解作为单应变换。这里，$\boldsymbol{A}=[\boldsymbol{a}_i]$ 是由每一组对应特征点的齐次坐标构成的矩阵，记为

$$\boldsymbol{a}_i=\begin{bmatrix}-\boldsymbol{x}_i & -\boldsymbol{y}_i & -1 & 0 & 0 & 0 & \boldsymbol{x}_i\widetilde{\boldsymbol{x}}_i & \boldsymbol{y}_i\widetilde{\boldsymbol{x}}_i & \widetilde{\boldsymbol{x}}_i\\ 0 & 0 & 0 & -\boldsymbol{x}_i & -\boldsymbol{y}_i & -1 & \boldsymbol{x}_i\widetilde{\boldsymbol{y}}_i & \boldsymbol{y}_i\widetilde{\boldsymbol{y}}_i & \widetilde{\boldsymbol{y}}_i\end{bmatrix}_{2\times 9} \tag{7.18}$$

其中，$\boldsymbol{x}_i=(\boldsymbol{x}_i,\boldsymbol{y}_i,1)$ 和 $\widetilde{\boldsymbol{x}}_i=(\widetilde{\boldsymbol{x}}_i,\widetilde{\boldsymbol{y}}_i,1)$ 是特征点的齐次坐标。为了避免函数优化过程中的平凡零解，需要添加约束条件 $\|\boldsymbol{h}\|=1$。这样通过对矩阵 $\boldsymbol{A}$ 进行奇异值分解，计算出符合特征点对齐误差最小的单应变换。

基于上述 DLT 求解方式，APAP 拼接方法采用移动直接线性变换方式（Moving DLT）计算网格每个格子对应的单应变换。假设 $\boldsymbol{x}_*$ 是一个网格顶点，那么该顶点对应的单应变换 $\boldsymbol{h}_*$ 应当是满足如下函数最小化的解：

$$\boldsymbol{h}_*=\underset{\boldsymbol{h}}{\operatorname{argmin}}\sum_{i=1}^{N}\|w_*^i\boldsymbol{a}_i\cdot\boldsymbol{h}\|^2,\quad w_*^i=\exp(-\|\boldsymbol{x}_*-\boldsymbol{x}_i\|^2/\sigma^2) \tag{7.19}$$

其中，权因子 w_*^i 定义了距离网格顶点不同远近的特征点对于待求解单应变换的影响，并通过预定义的方差 σ 进行控制。公式(7.21)仍可按照普通 DLT 的奇异值分解方式进行求解，但不同的格子是可以根据和特征点的距离调整相应的单应变换矩阵取值。在此基础上，得到的拼接图像具有更小的特征点对齐误差和扭曲。

例题 7-4　基于单应变换的图像拼接。

问题：对右图所示的两幅图像进行拼接，写出采用整体单应变换进行对齐的伪代码。

解答：假设两幅图像分别是 A 和 B，采用整体单应变换进行对齐得到拼接图像 S 的伪代码如下。

```
function homographyStitch (A, B, S)
    P_A ← featureDetect (A)                    /* 检测 A 中的特征点 */
    P_B ← featureDetect (B)                    /* 检测 B 中的特征点 */
    (M_A, M_B) ← featureMatch (P_A, P_B)       /* 匹配 A 和 B 中的特征点 */
    H ← calculate_matrix (M_A, M_B)            /* 根据匹配特征点对计算单应变换 */
    S ← A ∪ H · B
end function
```

□

7.4.2　形状保持的半单应变换图像拼接

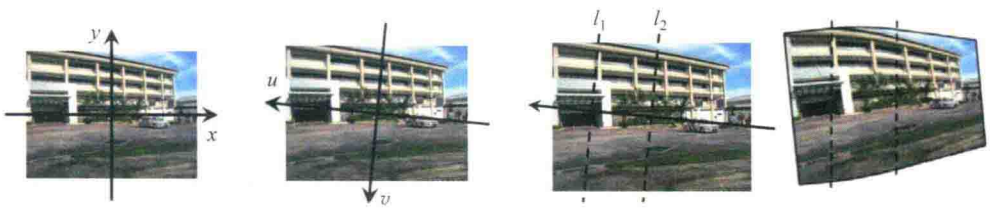
在图像拼接时，重合区域拼接的准确性主要由特征点对齐程度决定。而在非重合区域，则需要尽可能保持输入图像原来的内容不变。APAP 的方法虽然采用了局部单应变换减少扭曲，但是由于单应变换本身的局限性（例如直线的平行性无法保持），拼接后的图

像仍存在一定程度的形变。形状保持的半单应变换可以解决这些问题，其基本思想是在非重合区域采用形状保持的相似变换替代单应变换，因而称为形状保持的半单应变换方法(Shape-preserving Half-projective，SPHP)。该方法需要在重合区域和非重合区域之间建立过渡区域，实现从重合区域的单应变换渐变到非重合区域的相似变换的自然过渡。这样就能够确保使用两种不同性质的变换时，仍可以得到好的拼接图像。

为了得到从单应变换到相似变换的渐变，首先通过坐标系转换得到新的单应变换表达式。这个步骤目的是在新的 uv 坐标系中(如图 7.20(a)所示)，单应变换的表示形式可以简化为分母上只存在一个变量 u。具体来讲，对于匹配的特征点 $\boldsymbol{x}_i$ 和 $\widetilde{\boldsymbol{x}}_i$，坐标变换的过程如下：

$$\begin{aligned}\widetilde{x}_i &= \frac{h_1 x + h_2 y + h_3}{h_7 x + h_8 y + 1} = \frac{h_1 u + h_2 v + h_3}{1 - cu} \\ \widetilde{y}_i &= \frac{h_4 x + h_5 y + h_6}{h_7 x + h_8 y + 1} = \frac{h_4 u + h_5 y + h_6}{1 - cu}\end{aligned} \tag{7.20}$$

这样，u 增大，变换后的尺度变小；u 减小，变换后的尺度变大。而如果 u 是常数，那么单应变换退化为线性变换，实际上更接近于相似变换。因此，可以选择一个合适的 u 值，也就是对应于新坐标系上的一条分割线，将变换区域划分为单应变换和相似变换两部分区域(如图 7.20(b)所示)。



(a) 坐标变换　　(b) 分区域变换

图 7.20　SPHP 拼接中的变换(图片来自[81])

在 u 值所对应的分割线的左侧，采用单应变换 $\boldsymbol{H}$ 进行图像变换。而在该条分割线的右侧，采用相似变换 $\boldsymbol{S}$ 进行图像变换。前者保证了特征点对在对齐时的准确性，后者则减少变换后的扭曲。但是，这样会在边界处产生一个从单应变换到相似变换的突变，影响拼接效果。一个更自然的解决方法是寻找从单应变换到相似变换的渐变，也就是需要具有一定连续性的变化。这样在单应变换区域和相似变换区域之间，设计一个指定宽度的渐变区域，实现从单应变换到相似变换的连续过渡。通常情况下，该渐变区域定义为两条分割直线 l_1 和 l_2 之间的区域。那么，两幅图像最终的拼接变换 $\boldsymbol{W}(u,v)$ 表示为：

$$\boldsymbol{W}(u,v) = \begin{cases} \boldsymbol{H}(u,v), & R_H = \{u \leqslant u_{l_1}\} \\ \boldsymbol{T}(u,v), & R_T = \{u_{l_1} < u < u_{l_2}\} \\ \boldsymbol{S}(u,v), & R_S = \{u \geqslant u_{l_2}\} \end{cases} \tag{7.21}$$

其中，单应变换 $\boldsymbol{H}$ 可以采用 APAP 的方法进行计算，而相似变换 $\boldsymbol{S}$ 则根据匹配的特征点进行计算。在此基础上，过渡变换 $\boldsymbol{T}$ 则是根据单应变换 $\boldsymbol{H}$ 和相似变换 $\boldsymbol{S}$ 的线性插值来计算。最终，通过 $\boldsymbol{H}$、$\boldsymbol{T}$、$\boldsymbol{S}$ 三个变换作用到两幅图像中的不同区域，就得到最终的拼接结果。

7.4.3 自适应混合变换图像拼接

SPHP 方法虽然引入相似变换避免非重合区域的严重扭曲，但重合区域的单应变换仍有可能产生比较大的扭曲。为了结合单应变换和相似变换各自的优点，可以采用两种变换混合的方式计算图像拼接时的变换。具体来讲，通过局部单应变换和相似变换的混合，同时降低匹配特征点的对齐误差和形状扭曲。

从代数形式上看，相似变换是一种关于坐标的线性变换，而单应变换则是非线性变换。为了实现这两种变换在混合时的有效性，需要将单应变换进行线性化处理。这里通过泰勒展开公式的一阶近似，对 APAP 方法中每一点 $\boldsymbol{x}$ 处的单应变换 $\boldsymbol{h}$ 进行如下形式的近似表示：

$$\boldsymbol{h}(\boldsymbol{x}) \backsimeq \boldsymbol{h}(\boldsymbol{x}_0) + \boldsymbol{J}_h(\boldsymbol{x}_0) \cdot (\boldsymbol{x} - \boldsymbol{x}_0) \tag{7.22}$$

其中，$\boldsymbol{J}_h$ 是单应变换矩阵关于其元素的雅可比矩阵。在此基础上，将 APAP 方法中所采用的移动 DLT 求解中的单应矩阵改为公式(7.24)的形式进行计算，得到和网格的每个格子相对应的线性化单应变换。

在此基础上，在重合区域以特征点到两幅图像中心距离作为权值，对计算得到的变换进行线性混合，得到图像拼接的变形函数。相比于 APAP 和 SPHP，这种自适应混合变换拼接方法能够更好地减少拼接后图像内容的扭曲，如图 7.21 所示。

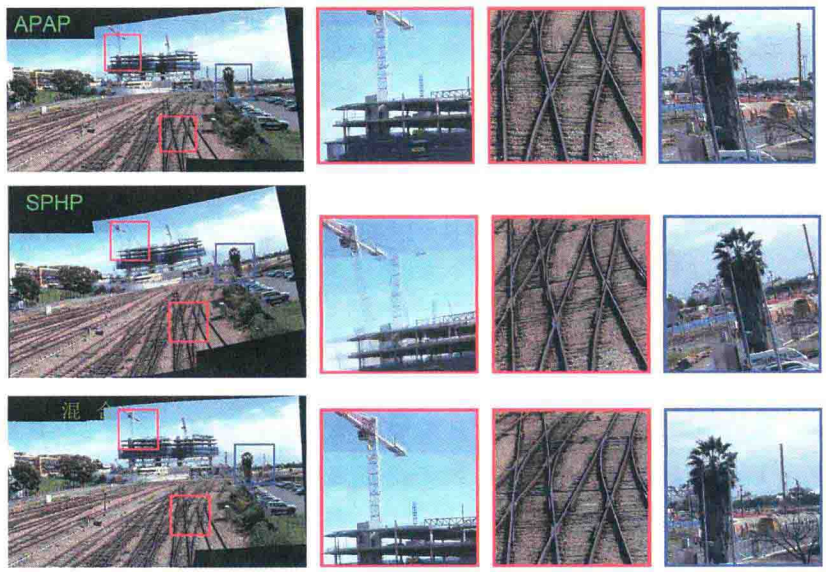


图 7.21 不同图像拼接方法的结果对比(图片来自[82])

7.4.4 视频拼接

与图像拼接相比，视频拼接更加困难。这里的难点主要体现在两个方面：首先是不同视角拍摄的视频存在视差，对应的帧图像在拼接时很容易产生重影现象；其次，拼接后的视频需要满足时空的连续性，也就是尽量减少新视频帧画面的跳跃。如果简单地逐帧

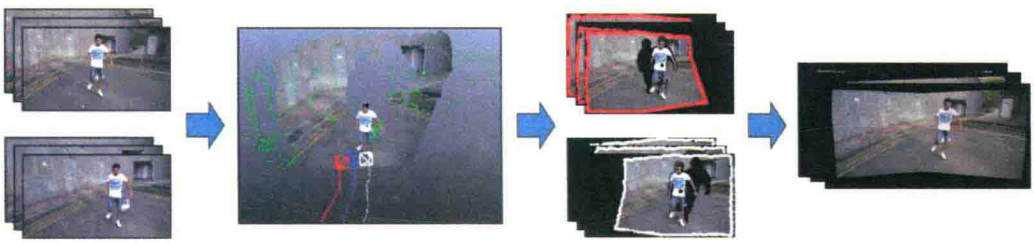
进行图像拼接，那么视频播放时就会产生比较明显的帧间跳跃。针对这两个问题，通常采用基于三维重建和基于内容保持的时空变形视频拼接。

1. 基于三维重建的视频拼接

这种方法基本思想是利用 3.4.2 节中基于视频的三维重建方法，获取三维空间相机拍摄的运动路径。如图 7.22 所示，该方法是对两条重建的路径进行融合，以此生成新的相机路径作为拼接后新视频的虚拟拍摄路径。在此基础上，通过三维空间到图像平面的投影来重构帧序列，生成拼接视频。其中，最关键的步骤是对相机路径进行融合和拼接视频帧序列重构。

在进行运动路径融合时，需要从输入的两个视频分别获得相应的相机运动路径。这表示为随时间变化的相机姿态序列，记为 $\boldsymbol{P}_l^t$ 和 $\boldsymbol{P}_r^t$。那么，采用简单线性加权平均的方式对两个相机姿态序列进行融合，从而得到拼接视频对应的虚拟运动路径，记为 $\boldsymbol{P}^t=(\boldsymbol{P}_l^t+\boldsymbol{P}_r^t)/2$。

根据拼接视频的运动路径，对两个视频的每一帧进行形状保持的变换。这样就可以在同一个坐标系下，将两幅变换后的帧画面组合为拼接后的视频帧。具体可以采用影像缩放中的图像变形方法（如 7.2.2 节所示），将两幅图像特征点映射到拼接后投影点的位置作为约束，生成拼接后的视频帧画面。



(a) 输入视频　(b) 路径融合　(c) 视频帧重构　(d) 拼接视频

图 7.22　基于三维重建的视频拼接（图片来自[83]）

2. 基于内容保持的时空变形视频拼接

这种方法基本思想是直接将图像拼接过程推广到视频帧序列，通过增加帧间连续性约束，实现时空一致的视频拼接。首先对输入的视频做整体对齐变换，作为拼接帧画面的初始状态，这里通常采用对应帧间变换的平均值进行整体的对齐。然后，构造局部内容保持的变形函数。该函数与影像缩放中图像变形函数的主要区别在于增加了相邻帧间网格顶点位置变化的时域连续性约束，以此实现对应帧的拼接。最后，根据变形结果完成对应帧的拼接。

此外，还可以借助三维的图割优化算法，寻找帧间连续的拼接线，以实现不同视频内容的无缝拼接，得到时空一致的新视频。

7.5　影 像 编 辑

影像编辑是指改变原来图像/视频的颜色、形状、结构、运动等信息，生成新形式的图像/视频。例如颜色迁移，可以通过一个模板图像改变图像原始的颜色；形状变形，是利用

局部几何形状的变化，改变图像中物体的形状。此外，通过编辑传播可以将图像中某一区域的更改，迅速传播到整个图像中其余区域，以此提高编辑效率。

影像编辑的一个基本要求是编辑后的图像内容和原图像具有一定的协调性。例如颜色迁移中编辑前后的脸部颜色要符合人类肤色的特点。对于视频而言，编辑后的结果要具有时空一致性，使得视频帧画面连续变化，否则也会产生画面跳跃等问题。

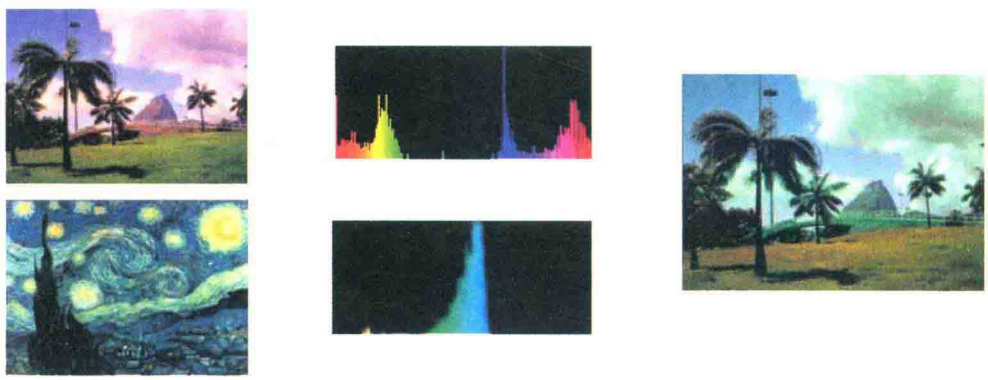
7.5.1 颜色迁移

颜色迁移使用用户提供的色彩作为模板，修正输入图像原始的颜色以满足模板色彩特点。为了使迁移后的颜色与图像自身的内容协调，通常使用基于特征匹配、基于笔画约束的颜色迁移方法。接下来具体介绍这两种方法。

1. 基于特征匹配的颜色迁移

这种方法主要思想是按照图像内容的连续性，在色相、色温等色彩特征空间进行迁移。这样可以使得迁移后的颜色与输入图像自身内容相吻合。具体来讲，如图 7.23 所示，根据模板图像和输入图像，分别计算各自的色相分布。色相是色彩的首要特征，是区别各种不同色彩的最准确的标准。从光学角度讲，色相差别是由光线波长的长短不同产生的。这样即便是同一类颜色，也能分为几种不同的色相。例如，黄颜色可以分为中黄、土黄、柠檬黄等。然后，对输入图像和模板图像进行色相划分，建立相应的直方图统计。那么，迁移过程就是通过二值匹配将输入图像的每个容器映射到模板图像对应的容器。最后，经过空间连续性处理，得到符合模板图像色相分布的迁移结果。

在迁移过程中，由于原图像直方图中距离比较近的直方块，在经过映射后距离变得过远，会产生伪边界效应，使得迁移后的结果协调性较差。这时可以通过直方块的合并策略更新映射，进一步去除伪边界效应。图 7.23(d)展示了基于特征匹配的颜色迁移结果。但是，这种方法有时候会产生一些语义上的错误，例如天空的颜色变成了黄色，从而不符合人对颜色感知的基本常识。



(a) 输入图像（上）和模板图像（下）　(b) 色相直方图　(c) 迁移结果

图 7.23 基于特征匹配的颜色迁移(图片来自[84])

2. 基于笔画约束的颜色迁移

这种方法通过用户交互的笔画颜色值作为模板，采用扩散的方式迁移至输入图像中其余的区域，完成笔画颜色的迁移。由于交互时引入了主观上的语义信息，这种迁移方式得到的结果更加自然。在进行颜色迁移时，灰度相近的相邻像素在迁移后应具有相近的颜色。因此，对于两个像素 $\boldsymbol{p}$ 和 $\boldsymbol{q}$，可以定义如下关于颜色 Y 的相似性：

$$w_{pq} = \mathrm{e}^{-(Y(\boldsymbol{p})-Y(\boldsymbol{q}))^2/2\sigma_p^2} \tag{7.23}$$

其中，σ_p 是图像颜色分布的标准方差。在此基础上，就可以把笔画颜色作为约束，建立关于迁移后像素颜色 U 的函数：

$$J(U) = \sum_{p}\left(U(\boldsymbol{p}) - \sum_{\boldsymbol{q}\in N(\boldsymbol{p})} w_{pq}U(\boldsymbol{q})\right)^2 \tag{7.24}$$

其中，$N(\boldsymbol{p})$ 表示像素 $\boldsymbol{p}$ 的局部邻域，通常假定为和当前像素直接相邻的 4 个像素。通过优化上述局部颜色相似性函数实现颜色迁移。图 7.24 展示了通过笔画的颜色实现整幅灰度图像的颜色迁移结果。在这个例子中，不同颜色的笔画实际上指定了各个区域在迁移后应该具有的颜色。

图 7.24　基于笔画约束的颜色迁移(图片来自[85])

7.5.2　图像变形

图像既可以看作像素域上的元素集合，也可以如同在影像缩放等处理中，作为嵌入网格的图形元素集合。这样，像素域和网格就成为图像的两种不同形式的定义域。图像变形是指通过操控图像定义域，实现图像整体或局部的几何形状改变。传统图像处理中的图像变形大多基于像素域表示，直接对每个像素进行操作。图形学中的图像变形则大多基于网格表示，通过改变网格的几何形状驱动图像中物体形状变形。相比于像素域的图像变形，基于网格的变形方法更加灵活，变形效果也更加多样。

移动最小二乘(Moving Least Squares，MLS)变形是一种典型的基于网格的图像变形方法。通过改变若干控制点的位置，驱动图像进行相应的变形。变形后的图像应满足控制点的几何位置约束。移动最小二乘变形是借助多重局部几何变换的加权作用，实现图像的整体变形，而在局部则需要满足控制点的位置约束。假设控制点 $\boldsymbol{p}_i$ 在图像变形后的位置记为 $\boldsymbol{q}_i$，那么对应于网格顶点 $\boldsymbol{v}$ 定义的变换为 $F_{\boldsymbol{v}}$，使得所有控制点在变换后的误差最小，记为

$$\underset{F_{\boldsymbol{v}}}{\operatorname{argmin}} \sum_i w_i \parallel F_{\boldsymbol{v}}(\boldsymbol{p}_i) - \boldsymbol{q}_i \parallel^2 \tag{7.25}$$

其中，w_i 表示控制点到网格顶点距离对变换的影响，通常定义为 $w_i = 1/|\boldsymbol{p}_i - \boldsymbol{v}|^{2\alpha}$，$\alpha$ 则是

被设置成固定的常值。

由公式(7.25)定义的图像变形函数具有两个重要的性质：局部性，即不同网格局部的变形函数是不同的；连续性，即图像的变形效果在相近空间位置上的相似性。对于常见的仿射变换、相似变换和刚体变换，都可以根据公式(7.27)计算出具有显式表达式的变形函数。

具体来讲，对于仿射变换 $F_v(\boldsymbol{p}_i)=\boldsymbol{p}_i\cdot\boldsymbol{M}+\boldsymbol{T}$，其中 $\boldsymbol{M}$ 是一个线性变换，$\boldsymbol{T}$ 表示平移。根据变形前后特征点的位置变化，可以得到相应的几何中心位置的变化，分别记为 $\boldsymbol{p}_*=\sum_i w_i\boldsymbol{p}_i\Big/\sum_i w_i$ 和 $\boldsymbol{q}_*=\sum_i w_i\boldsymbol{q}_i\Big/\sum_i w_i$。在此基础上，得到如下仿射变换的表达式：

$$\boldsymbol{M}=\Big(\sum_i \hat{\boldsymbol{p}}_i^{\mathrm{T}}\cdot w_i\,\hat{\boldsymbol{p}}_i\Big)^{-1}\sum_j w_j\,\hat{\boldsymbol{p}}_j^{\mathrm{T}}\cdot\hat{\boldsymbol{q}}_j,\quad \boldsymbol{T}=\boldsymbol{q}_*-\boldsymbol{p}_*\cdot\boldsymbol{M} \tag{7.26}$$

这里，$\hat{\boldsymbol{p}}_i=\boldsymbol{p}_i-\boldsymbol{p}_*$ 和 $\hat{\boldsymbol{q}}_j=\boldsymbol{q}_j-\boldsymbol{q}_*$ 分别表示以几何中心为原点的坐标系下的控制点位置。

对于相似变换，$\boldsymbol{M}$ 的表达式是

$$\boldsymbol{M}=\frac{1}{\mu_s}\sum_i w_i\begin{pmatrix}\hat{\boldsymbol{p}}_i\\ -\hat{\boldsymbol{p}}_i^{\perp}\end{pmatrix}(\hat{\boldsymbol{q}}_i^{\mathrm{T}},-\hat{\boldsymbol{q}}_i^{\perp\mathrm{T}}) \tag{7.27}$$

其中，$\mu_s=\sum_i \hat{\boldsymbol{p}}_i\cdot\hat{\boldsymbol{p}}_i^{\mathrm{T}}$，$\hat{\boldsymbol{p}}_i^{\perp}$ 和 $\hat{\boldsymbol{q}}_i^{\perp}$ 分别表示对向量 $\hat{\boldsymbol{p}}_i$ 和 $\hat{\boldsymbol{q}}_i$ 的正交操作，也就是说该操作应满足 $(x,y)^{\perp}=(-y,x)$，平移 $\boldsymbol{T}$ 具有和公式(7.26)相同的形式。。

对于刚体变换，$\boldsymbol{M}$ 的表达式是

$$\boldsymbol{M}=\frac{\sum_i w_i\begin{pmatrix}\hat{\boldsymbol{p}}_i\\ \hat{\boldsymbol{p}}_i^{\perp}\end{pmatrix}(\hat{\boldsymbol{q}}_i^{\mathrm{T}},\hat{q}_i^{\perp\mathrm{T}})}{\sqrt{\Big(\sum_i w_i\,\hat{\boldsymbol{q}}_i\cdot\hat{\boldsymbol{p}}_i^{\mathrm{T}}\Big)^2+\Big(\sum_i w_i\,\hat{\boldsymbol{q}}_i\cdot\hat{\boldsymbol{p}}_i^{\perp\mathrm{T}}\Big)^2}} \tag{7.28}$$

刚体变换的平移分量 $\boldsymbol{T}$ 具有和公式(7.26)相同的形式。

图 7.25 展示了使用相同控制点约束的三种变换形式的变形结果。可以看到，刚体变换在同样的控制点条件下能够更好地保持局部几何细节特征，具有最佳的变形效果。

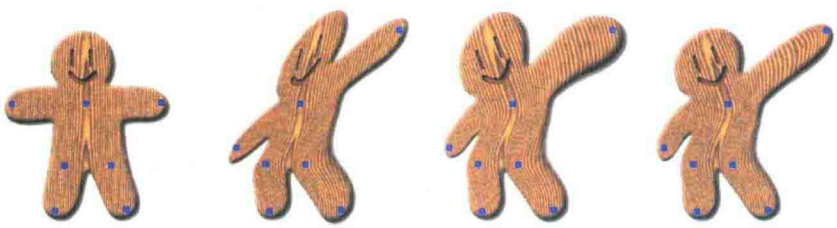


图 7.25 移动最小二乘图像变形(图片来自[86])

7.5.3 编辑传播

图像编辑时通常需要人工交互来指定编辑意图，然后利用相应的方法实现整幅图像上颜色、形状等的改变。为了提高编辑的效率，有些图像编辑方法会充分利用像素域的位

置、颜色等属性的相似性，通过传播特定区域人工交互的编辑效果，实现整幅图像的快速编辑。

编辑传播的基本原则是相似的物体或区域在编辑后应当具有相近的编辑效果。因此，首先需要定义像素之间的相似性。该相似性既包括空间位置的相似性，也包括像素颜色等表观的相似性。常用的相似性函数定义为

$$s(\boldsymbol{p}_i, \boldsymbol{p}_j) = \exp(-\| \boldsymbol{c}_i - \boldsymbol{c}_j \|^2/\sigma_c)\exp(-\| \boldsymbol{x}_i - \boldsymbol{x}_j \|^2/\sigma_x) \tag{7.29}$$

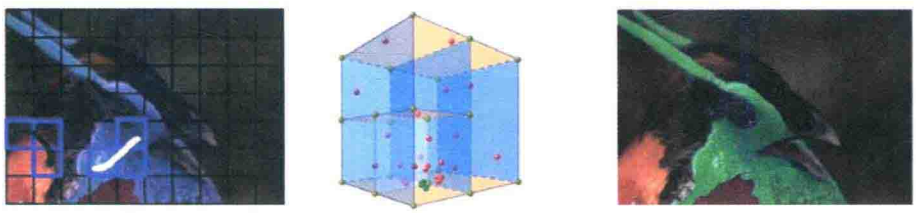
其中，$\boldsymbol{c}_i$ 和 $\boldsymbol{c}_j$ 分别表示像素 $\boldsymbol{p}_i$ 和 $\boldsymbol{p}_j$ 的颜色，$\boldsymbol{x}_i$ 和 $\boldsymbol{x}_j$ 分别表示像素的位置，σ_c 和 σ_x 则是像素颜色和位置各自分布的标准方差。

假设通过人工交互的方式进行编辑的效果，是在区域 Ω 内将像素颜色改变为指定的颜色，记为 $\{\boldsymbol{c}_i = g_i \mid \boldsymbol{p}_i \in \Omega\}$，那么整幅图像在编辑后的颜色变化应满足如下函数的最优解：

$$\sum_i \sum_j s_{ij} (\boldsymbol{c}_i - \boldsymbol{g}_i)^2 + \lambda \sum_i \sum_j s_{ij} (\boldsymbol{c}_i - \boldsymbol{c}_j)^2 \tag{7.30}$$

这里第一项是交互编辑约束项，指定了相应像素应该满足的编辑效果；第二项是连续性约束项，使得人工编辑传播至整个图像的同时，仍能具有一定的空间连续性。

然而，上述编辑传播函数是非线性函数，直接进行优化的效率较低。为了提高计算速度，可以采用 K-D 树的数据结构对特征空间做自适应划分，如图 7.26(a)所示。借助这种有效的高维空间划分方式，可以加快公式(7.30)所示的编辑传播函数中相关参数的查找速度，从而提高编辑传播的效率。



(a) K-D 树传播编辑　　(b) 颜色编辑传播结果

图 7.26　基于 K-D 树加速的编辑传播(图片来自[87])

具体来讲，首先将输入图像映射到特征空间。该特征空间是由像素位置 (x_i, y_i) 及其颜色的 R、G、B 三个通道分量构成的五维空间。然后，对特征空间进行 K-D 树划分，建立层次组织结构，同时也将图像的每个像素映射成高维空间中的点。此外，人工交互区域的像素也映射到高维特征空间。接着，利用 K-D 树节点插值，计算其他非交互区域点的数值，从而快速生成所有特征空间点在编辑后的结果。最后，把那些在特征空间改变后的数值映射回图像空间，得到该图像编辑后的结果。因为图像空间中相似的编辑在特征空间具有自然的近似关系，所以能够大大提高编辑效果的传播效率。在图 7.26(b)所示的例子中，仅需要两个笔画的交互，就可以将颜色编辑的效果快速传播到其他非交互区域。

7.5.4 视频去抖

视频帧间画面的运动是视频区别于单幅静态图像的重要属性。移动端摄像机在运动环境下拍摄的视频往往存在画面抖动。这种抖动现象一方面影响了用户对视频内容的正常观看，另一方面也会增加视频编码时的码率，影响视频的存储和传输。视频去抖动就是借助硬件或者算法处理视频帧序列，使得运动环境下拍摄的视频仍能够具有平稳的帧间运动。因此，视频去抖一般可以分为硬件去抖和数字去抖两种类型。

硬件去抖是利用特殊的稳定装置对运动中的摄像机加以固定，例如图 7.27 中所示的三脚架、导轨、云台等。这样就可以补偿摄像机的抖动，进而能够拍摄出稳定的视频。硬件去抖虽然效果明显，但是装置成本较高、体积较大，并不适合于普通用户的便携式使用。另一种方式是对拍摄的视频帧画面进行数字图像变形处理，或者再结合移动端上的陀螺仪等传感器的物理数据，使得变形后的视频帧序列能够满足平稳运动。这种数字去抖方式不需要额外的硬件装置，成本较低，而且使用方便，因而被广泛用于视频去抖。

图 7.27 硬件稳定装置：三脚架、导轨、云台

数字去抖一般分为运动估计、运动平滑和运动补偿三个步骤。运动估计是从拍摄的视频帧或者传感器读取的图像或者物理数据估计帧间运动，进而建立起描述帧间变化的运动模型。运动平滑是对帧间运动进行平滑处理，去除画面抖动成分，得到帧间平稳的运动。运动补偿则是对原始帧进行图像变形，使得变形后的帧序列满足运动平滑后的平稳运动状态。这其中的运动估计和运动平滑是视频去抖的关键步骤，直接影响了去抖效果。

按照运动模型所描述时空维度的不同，数字视频去抖方法大致可以分为三类：二维方法、三维方法和 2.5 维方法。这里，二维方法将视频运动描述成二维平面内的运动，需要从帧间变化估计相应的二维运动；三维方法将视频运动直接表示为摄像机在三维空间的运动，需要从视频帧序列恢复三维空间信息；2.5 维方法介于二维方法和三维方法之间，利用部分三维空间信息补充二维方法的运动模型。接下来介绍这三类视频去抖方法。

1. 二维方法

视频是三维空间投影到二维成像平面形成的连续序列，因此可以通过帧间光流或者几何变换，将视频运动描述为图像平面内的运动。光流对应于像素位移形成的图像局部运动，而几何变换则是图像整体运动。常用的几何变换包括相似变换、仿射变换、单应变换等，都可以按照影像拼接时基于匹配特征点的方式计算帧间几何变换。

以几何变换描述的运动模型为例，假设相邻两帧 $\boldsymbol{I}_{k-1}$ 和 $\boldsymbol{I}_k$ 之间的几何变换记为 $\boldsymbol{G}_k$，那么如图 7.28 所示，视频运动路径 $\boldsymbol{P}_k$ 就可以表示为几何变换的累积形式：$\boldsymbol{P}_k = \boldsymbol{I} \cdot \boldsymbol{G}_2 \cdots$

$\boldsymbol{G}_k$。它实际上根据帧间变化描述了从起始帧到当前帧的运动状态。在此基础上，就可以通过寻找平稳的帧间运动，达到去除视频原有抖动的目标。最简单的方法是将运动路径 $\boldsymbol{P}_k$ 看作随时间变化的信号，那么可以利用低通滤波对运动路径进行平滑。以高斯滤波为例，平滑后的运动路径$\widetilde{\boldsymbol{P}}_k$ 可以通过如下表达式来计算：

$$\widetilde{\boldsymbol{P}}_k = \sum_{i \in N_k} \boldsymbol{P}_i \otimes \mathrm{e}^{-k^2/2\sigma^2} / \sqrt{2\pi\sigma^2} \tag{7.31}$$

其中，N_k 表示中心位置在第 k 帧的窗口，通常选取前后各 25 帧组成，而符号⊗表示卷积操作，标准方差 σ 一般设置为$\sqrt{k}$。为了满足平滑后的运动路径，需要对每一帧进行变形以补偿原始路径和平滑路径之间的差异。假设对视频帧 I_k 进行补偿的几何变换是 $\boldsymbol{S}_k$，那么它应该满足 $\boldsymbol{S}_k = \boldsymbol{P}_k^{-1} \cdot \widetilde{\boldsymbol{P}}_k$。这样对每一帧按照几何变换 $\boldsymbol{G}_k$ 做变形，得到的新视频就能够具有平稳的运动。如果使用光流运动模型，也可以采用高斯滤波对帧间像素位移进行平滑，再补偿原始位移和平滑位移后，就可以得到运动平稳的视频。

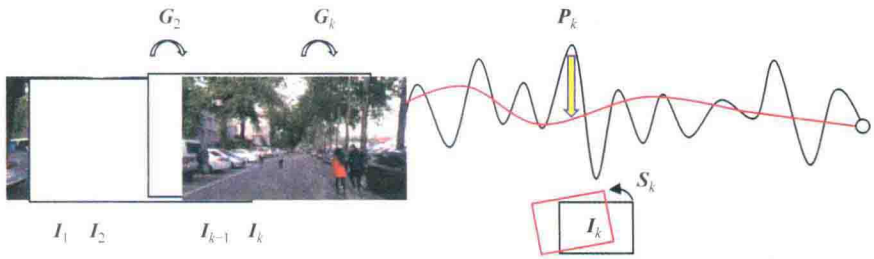


图 7.28　基于帧间几何变换的运动估计、运动平滑和运动补偿

二维方法计算简单，处理效率高，大多数情况下能够对视频进行实时去抖。但是由于缺乏三维空间信息，可能会影响视频去抖后的视觉观看效果。这里的本质原因是它将视频运动简化为平面上的二维运动，导致部分信息的缺失。

2. 三维方法

由于视频是记录三维空间的运动，最理想的视频运动描述应当是能够反映摄像机在三维空间的运动情况。根据第 3 章介绍的三维重建方法，可以从视频帧反向重建拍摄场景的三维模型，以及摄像机在三维空间的运动路径。在此基础上，就可以先对三维运动路径做平滑处理，然后按照平滑路径的投影变换对原始视频帧做变形，从而获得满足该平滑路径的视频帧序列。

因为三维方法可获得完整的三维运动路径，所以它能够更准确地建立三维空间中的运动模型，由此进行视频去抖的效果更好。但是，正如第 3 章中介绍的各种三维重建方法，它们往往算法复杂度高、计算量大，很难对视频去抖进行实时处理。

3. 2.5 维方法

针对二维方法计算效率高、三维方法去抖效果好的情况，2.5 维方法汲取了二维方法和三维方法各自的优势，能够更好地用于视频去抖。简单来讲，2.5 维方法仍然使用帧间光流或者几何变换描述视频运动，但是在运动平滑过程中引入适当的三维空间关系作为约束，使得平滑过程中的帧间运动能够符合三维运动属性。这样既能发挥二维方法的效

率优势，也能具有一定的三维方法的去抖效果。

最简单的 2.5 维方法是采用分区域几何变换来代替二维方法中的整体几何变换，从而体现出不同区域的运动差异。具体来讲，视频的每一帧都按相同的规则划分为四边形网格，也就是类似图像缩放时的嵌入网格。然后，相邻两帧之间的几何变换则根据对应的网格区域分别进行计算，形成空间中的多条运动路径。接下来，对于每一条运动路径进行平滑处理，例如采用高斯平滑，得到相应的平稳路径。最后，通过网格驱动的图像变形，对视频帧进行运动补偿，得到最终的平稳视频。

其他一些 2.5 维方法则借助多视角视觉中的几何关系作为约束，例如极线约束，提高平滑处理时的去抖效果。所谓极线(Epipolar)，是指三维空间中的点投影到成像平面上时形成的投影直线。极线约束则是指同一点在不同视角下投影时应满足共线性。事实上，极线约束在第 3 章中基于图像的三维重建中也被使用(参见图 3.18)。利用这种极线约束来限制平滑处理时的补偿变换，使得去抖动后的视频帧仍满足该约束条件。这样就避免了三维重建来引入三维空间约束，是一种更经济而有效的数字去抖策略。

7.6 小　　结

传统影像处理方法是在像素域进行计算。这种像素级别的图像/视频处理的灵活性和有效性受到限制。图形学方法则可以开展从亚像素到网格区域的多级别处理，借助几何优化提高图像/视频处理的灵活性和有效性，在影像抠图、缩放、融合、拼接、编辑等问题都取得了更优的效果。

然而，目前的图形学方法大多需要借助手工交互提供额外的约束，以保持影像处理的语义有效性。如何通过数据挖掘、机器学习的方法更高效地提取语义信息，并能用于提高影像处理的有效性，是当前基于图形学方法进行影像处理的重要问题之一。

思考题

7.1　图像分割和图像抠图的区别是什么？各自有什么样的应用？

7.2　图像抠图中的交互方式有哪些？不同方式的代表性抠图方法有哪些？

7.3　适用于不同尺寸显示屏的图像缩放方法有哪些？各自的优缺点是什么？

7.4　图像拼接时重叠区域和非重叠区域各自需要解决哪些问题，才能得到好的拼接结果？

7.5　采用移动最小二乘进行图像变形的数学原理是什么？采用不同形式的几何变换的变形效果有什么区别？

7.6　图像编辑方法推广到视频时的难点是什么？常用的解决手段有哪些？

7.7　数字视频去抖的常见方法有哪几种类型？各自的优缺点是什么？

7.8　基于图形的影像处理方法有什么特点？它比传统图像/视频处理方法有哪些优势？

第 8 章 计算摄像

数码相机的普及使得摄像变得越来越容易，同时通过计算机进行数字图像处理也更为便捷。然而，普通数码相机的光学和电子器件构造及其成像机理决定了影像获取的能力和质量。这在一定程度上限制了图像后期处理的效果。计算摄像(Computational Photography)是一种通过光学编码和计算解码获取影像并进行图像处理的手段。计算摄像扩展了传统成像、数字成像及图像处理的能力，提高了影像获取的质量。

本章介绍计算摄像相关的基础知识。首先根据摄像学的发展，介绍传统摄像、数字摄像和计算摄像的基本概念、光学原理和涉及的图像处理算法。然后针对计算摄像的实际用途，介绍计算光场成像和计算光谱成像这两类典型计算摄像的应用。

8.1 摄像学的发展

摄像起源于战国时期的小孔成像。战国初期，墨子和他的弟子们完成了世界上第一个小孔成像的实验，并通过文字记录在《墨经》中："景到，在午有端，与景长，说在端。"这句话解释了小孔形成倒像的原因，进而指出了光沿直线传播的性质。这是对光直线传播的第一次科学解释。而西方世界直到公元前 350 年，古希腊学者亚里士多德才提出基本的光学法则，从此了解小孔成像的光学原理。事实上，小孔成像也成为后来照相机的基本工作原理。简单来讲，它将真实世界中的场景通过光线的各种传播特性准确地投射到具有光线强度记录能力的成像介质平面上，从而获得符合用户要求的影像画面，最后再通过不同的存储介质保存下来。

按照成像方式的不同，摄像学的发展经历了传统摄像、数字摄像和计算摄像三个阶段。不同发展阶段的成像介质元器件、对焦方式、曝光方式等也发生了较大的变化。如图 8.1 所示，传统摄像主要是指数码相机出现之前的摄像技术，典型的以胶卷作为成像元器件；数字摄像是数码相机出现以后的摄像技术，典型的以电荷耦合器件(CCD)和互补金属氧化物半导体(CMOS)作为成像元器件；计算摄像则是在数字摄像基础之上的摄像技术，尤其是以光场相机为代表的新型成像方式的出现，大大扩展了以往摄像的能力和质量。

8.1.1 成像设备

随着光学系统、成像介质等的发展，摄像设备和摄像技术也产生了巨大的变革，经历了从暗箱写生到银版照相、胶卷相机、数码相机，再到手机相机、光场相机等的发展历程，

	小孔成像	暗箱写生	银版照相	胶卷相机	数码相机	手机拍摄	光场相机
光学	小孔	小孔	凸透镜	镜头组	镜头组	微镜头组	微镜头阵列
介质	光屏	纸张	金属	胶卷	CCD	CMOS	CCD
对焦	滑动标尺	拉箱	拉箱	手动	马达	马达	马达

图 8.1 摄像学的发展阶段

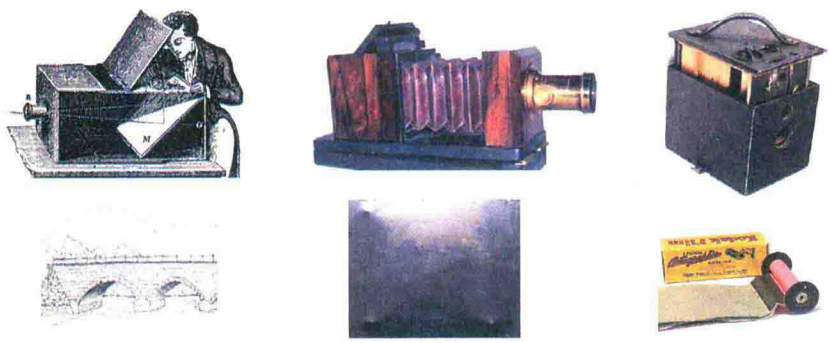
深刻影响了人们的日常生活。接下来介绍传统摄像、数字摄像和计算摄像这三个阶段的简单发展历程。

1. 传统摄像

传统摄像主要包括暗箱写生、银版照相、胶卷照相等阶段。传统摄像是通过模拟方式，借助化学介质记录光线的强度信息，从而完成拍摄场景的成像。因此，传统摄像也称为模拟摄像。

15 世纪末期，出现了基于小孔成像原理制作的暗箱(Camera Obscure)，成为现代照相机的雏形(如图 8.2(a)所示)。利用这种工具，人们只要将影像反射在画纸上，然后用铅笔描绘出轮廓再上色，就可以完成一幅很有真实感且完全符合真实比例的画像。这种方式实际上是利用暗箱提供拍摄场景中物体成像的一个参考，最终形成画像还是靠人通过手绘写生的方式在放置于暗箱之中的画板上来完成。

1822 年，法国人涅普斯(Joseph Nicéphore Niépce)在感光材料上制出了世界上第一张照片。1838 年，法国物理学家达盖尔(Louis Daguerre)发明了银版照相法(如图 8.2(b)所示)，该方法是利用镀有碘化银的钢板在暗箱里曝光，然后以水银蒸汽显影，再辅以普通食盐定影。1839 年 8 月 19 日，法国政府宣布放弃对银版摄影术这项发明专利的拥有权，并将其公之于众。人们通常以这一天作为摄像学的开端。



(a) 暗箱相机　(b) 银版相机　(c) 胶卷相机

图 8.2 早期的传统摄像设备及介质

1888年，美国Kodak公司生产出一种新型感光材料——胶卷。它是一种柔软、可卷绕的介质，是在聚乙酸酯片基上涂抹卤化银来制作。这是成像感光材料一个质的飞跃。同年，该公司推出了世界上第一台安装胶卷的便携式方箱照相机Kodak No.1，成为现代消费型相机的始祖（如图8.2(c)所示）。

采用胶卷的传统摄像设备可以记录非常真实的场景画面，而且成像质量非常高。但其缺点也很明显，也就是无法即时成像，需要在暗房冲洗定影才能看到拍摄的场景画面。此外，胶卷记录的影像在后期编辑时也很困难，很难进行颜色、亮度等属性的修改，而且其使用的化学成分也不利于照片的长久保存。

2. 数字摄像

数字摄像利用电子传感器把光学信号转换成电子数据进行影像的存储，也就是通过光电转换记录拍摄场景的影像。1969年，电荷耦合器件（CCD）芯片作为相机感光材料在美国阿波罗登月飞船上搭载的照相机中得到应用。如图8.3(a)所示，CCD是通过布置微小光敏物质作为成像像素的一种半导体器件，能够把光学信号转化为电信号，从而为照相感光材料的电子化和数字化打下了技术基础。1981年，日本Sony公司经过多年研究，生产出了世界第一款采用CCD做感光材料的数码相机Mavica，开启了电子传感器替代胶片作为成像元器件的时代，使传统摄像变成了数字摄像（如图8.3(b)所示）。

除了CCD外，互补金属氧化物半导体（CMOS）也是另一种常用的感光介质。1987年，日本CASIO公司推出了采用CMOS芯片作为感光材料的相机。CMOS通过外界光线照射像素阵列，发生光电效应，在像素单元内产生相应的电荷把光学影像转化为电信号。CMOS具有比CCD更低廉的成本，往往用于手机等移动设备上的摄像机。这也使得数字摄像的应用更加普及。

(a) CCD和CMOS

(b) Mavica数码相机

(c) J-SH04手机相机

图8.3 数字成像元器件及设备

事实上，随着智能手机等移动设备的大众化，手机相机也迎来了快速发展时期。2000年，日本SHARP公司发布了内置11万像素CCD摄像头的J-SH04手机（如图8.3(c)所示），这是世界上第一款照相手机，大大革新了手机的功能。2003年，该公司又推出了J-SH53手机，可拍摄最大1144×858像素的照片，成为世界上第一款百万像素级别的照相手机。

简单来讲，数字摄像也是基于小孔成像原理，但能够实现对拍摄影像的即时预览，也就是拍摄完后能够马上浏览照片。同时，数字摄像拍摄的照片更容易进行后期编辑和保存，尤其是结合计算机进行图像处理。这就大大提高了照片的质量和保存时间。但是受

视场角、曝光时间等限制，数字摄像和传统摄像都是仅记录简单透镜阵列线性投射的光线信息，例如光线强度或色彩，在使用时仍然受限。

3. 计算摄像

计算摄像，还是采用和数字摄像类似的成像介质，但是在数字摄像的基础上，通过改变成像光路或者成像模式，以及配合更加高效和定制的图像处理算法，进一步提高了成像能力和影像质量。2005 年，Stanford 大学 Ren Ng 博士开发了第一款手持式光场相机，并推出了面向普通消费者的产品。光场相机实现了先拍照后对焦的成像功能，一定程度上改变了以往的成像方式。这也是计算摄像技术最成功的应用之一。图 8.4 展示了第一代和第二代光场相机。

图 8.4 第一代和第二代光场相机

计算摄像通过将光学成像和计算机图像处理相结合，能够记录更高空间分辨率、角度分辨率的光学信息。此外，还可以记录可见光以外谱段的光学信息。因此，通过计算摄像可以获取更高维度、更宽尺度、更大深度的场景影像，丰富了人们获取真实世界信息的手段。2016 年，SIGGRAPH 技术成就奖获得者、MIT 大学教授 Frédo Durand 认为“计算成为一种新形式的光学器件”。计算摄像也为摄像学提供了一个全新的发展模式，为拓展摄像功能和提升摄像质量提供了新的途径。

8.1.2 成像原理

光学成像的过程是通过透镜采集从场景投射的光线，然后通过感光介质记录真实世界场景的影像。但是，成像时却不能直接将感光介质暴露在光线中，这样会造成在摄像时介质上每一点记录物体上所有点的信息，导致介质上每一点的颜色相同，达不到物体成像的效果。

为了能够在感光介质上显现出想要的物体影像效果，需要对到达感光介质的光线进行约束。小孔成像是最早采用的约束手段。如图 8.5 所示，它是基于光的直线传播，在介质上收集投射过来的光线信息。小孔成像的特点是不存在几何扭曲，体现在直线成像后仍是直线。同时，小孔成像具有无限景深，也就是每一点会都在介质平面上成像，因此理论上都可以清晰成像。

小孔成像的主要问题是孔径的设置。由于光的波动性，孔径越小，越容易产生衍射，造成影像产生几何模糊、亮度低等问题；而大孔径就会产生明显的马赛克效应，降低影像清晰程度。因此，直接的小孔成像很难形成高质量的影像。

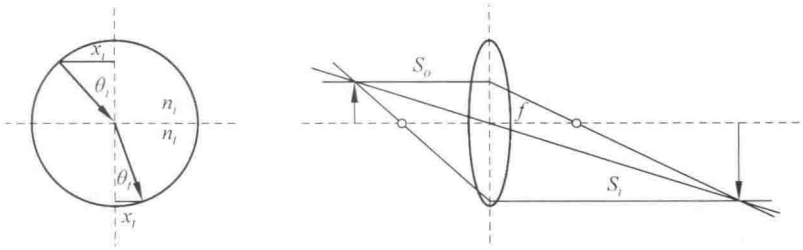
另外一种约束光线传播的手段是在光路中使用透镜，称为透镜成像。这种方式一方面由于透镜的汇聚作用会增加成像光线的强度；另一方面也可以有效地避免几何扭曲和

图 8.5 小孔成像与透镜成像

模糊等现象(如图 8.5 所示)。简单来讲,透镜成像是利用了光的折射现象。这种现象是指当光线从一种介质进入另一种介质时,由于传播速度的不同,在介质表面发生传播方向的偏转。物理上,光的折射遵循如下斯涅耳定律:

$$n_i \cdot \sin\theta_i = n_t \cdot \sin\theta_t \tag{8.1}$$

其中,n_i 和 n_t 是两种介质的折射率,θ_i 和 θ_t 分别表示入射角和折射角(如图 8.6(a)所示)。因此,通过透镜可以起到汇聚光线的作用。



(a) 斯涅耳定律 (b) 球面透镜成像

图 8.6 基于光线折射的球面透镜成像

在选择透镜时,双曲透镜是能够将平行光线汇聚到一个聚焦点的理想透镜,也是透镜成像的最佳选择。但是双曲透镜制作工艺复杂,加工困难。球面透镜也能起到汇聚光线作用,而且加工简单。常用的透镜是两个球面透镜的组合,也就是两个球面相交构成的薄透镜。当球面半径固定时,通过这种透镜成像的物距、像距和焦距满足如下高斯公式:

$$\frac{1}{s_o} + \frac{1}{s_i} = \frac{1}{f} \tag{8.2}$$

其中,s_o 和 s_i 分别是物距和像距,f 是透镜的焦距(如图 8.6(b)所示)。这里,物距是拍摄物体上的物点到透镜的距离,像距是该点发出的光线经透镜成像的像点到透镜的距离,焦距则是透镜焦点到透镜的距离。

从公式(8.2)所示的高斯公式可以看出,对于平行光,也就是物距 s_o 趋向于无穷大时,经过透镜后汇聚到焦点位置。这时,像距等于焦距,无限远处的物体成像为一个点。此外,经过透镜中心的光线的方向不会发生改变,而来自于和透镜平行平面的光线则会聚焦在平行于透镜的另一个平面。但是球面透镜通常存在球差现象,也就是当球面孔径较大时,位于光轴上的物点所发出的光线,经球面折射后不再交于一点,由此形成球面像差。在实际相机里面,通常采用镜头的组合来消除或削弱球差,这使得透镜成像能够获得高质量的影像。

8.1.3 相机成像

单一透镜成像时除了产生像差，还会引起色差等问题。所谓色差，就是经过透镜所形成的像改变了物体原有的颜色。这实际上也是由透镜对不同波长的光的折射程度不同所造成的。因此，如前所述，实际的相机镜头是由一系列的透镜组合而成，这样就能最大限度地降低成像时的像差、色差等问题，在成像介质上形成最佳的影像。此外，采用多个透镜还能够方便调焦，适合于拍摄不同远近的场景。除了透镜外，相机成像效果还和曝光程度、景深等因素密切相关。接下来简单介绍和透镜成像相关的几个概念。

1. 曝光程度

曝光程度，由进入镜头光线的辐照度和曝光时间所决定。辐照度，也就是单位时间内的光通量，由光圈控制其大小。曝光时间，则由快门打开时间所决定。光圈是一个用来控制光线透过相机透镜，进入机身内感光介质平面光量的装置。光圈大小通常使用光圈数表示，也就是焦距和孔径的比值。例如，对于焦距为 50mm 的透镜，光圈数 $f/2.0$ 表示光圈的孔径是 25mm，其中 f 是焦距。一般来讲，光圈数越小，能够接收的辐照度越多，成像时转化成的数字信号越强。一般普通相机的最小光圈数是 0.5，而单反相机的最小光圈数是 1.0。

快门的作用是控制光线照射感光材料的持续时间，主要有叶片式和焦平面式两种类型。叶片式快门也叫镜间快门。它由一系列薄钢叶片组成，放置在镜头的各个单元之间。这种快门使用时安静，但速度慢、价格高。焦平面式快门摆放在胶片或 CCD 前面，声音比较大，但速度快、价格低。然而，由于焦平面快门大多采用卷帘方式运转，有可能会产生运动扭曲。快门的速度实际上就控制了感光介质曝光的时间。采用 1 秒的比例作为单位，最短曝光时间一般要控制为 $1/f$。例如，焦距 500mm 的透镜，快门速度应控制在 1/500 秒，这样才能在接收的光线强度和成像质量之间达到最优的平衡。

2. 景深

景深是指在镜头焦点前后位于容许弥散圆范围内的景物的深度。理论上，只有像点位于成像平面上的景物在成像后才能汇聚到一点，称为合焦。否则，景物上的点在成像平面上会变成一个散焦光斑，称为弥散圆(Circle of confusion)。相机成像后，人眼对弥散圆大小的感知，也与感光元器件、观看距离等有关系。例如 35mm 胶片，人眼能感受到的弥散圆大小通常为 0.02mm。超出弥散圆范围的成像，就形成了散焦模糊(Defocus blur)。因此，弥散圆通常被认为是定义了影像清晰和模糊的界限，对于评价成像质量具有重要的参考意义。

一般而言，景深与光圈大小成反比关系，也就是说在容许弥散圆大小下，光圈越大，景深越小；光圈越小，景深越大。景深与焦距成反比关系，也就是说镜头焦距越长，景深越小；焦距越短，景深越大。景深与物距成正比关系，也就是说景物离镜头距离越远，景深越大；距离越近，景深越小。因此，在实际拍摄过程中，往往需要有一定的经验积累才能很好地控制这些因素，获得高质量的影像。

8.2 数字摄像

CCD和CMOS等电子传感器的出现，使得数字摄像逐渐取代了传统摄像。数码相机，就是利用电子传感器把光学影像转换成电子数据，进而实现数字摄像的照相机。在数码相机中，为了获取更高质量的影像，在数字摄像时会使用图像信号处理器(Image Signal Processing，ISP)对采集的图像信号进行预处理。这很大程度上也决定了最终成像的质量。事实上，ISP还起到适配不同厂商生产的传感器的作用，也就是通过后期ISP的数字校正，对传感器的差异进行修正，使其都能够拍摄到理想的影像。目前，数码相机ISP所具备的常见功能有去马赛克、3A调整、白平衡、去噪/锐化、压缩等。这些也是数字图像处理中常见的问题。接下来具体介绍这方面的相关知识。

8.2.1 去马赛克

物理上，CCD只会感应光线的强度，而无法区分不同的颜色。因此，需要选择合适的色彩调制机制，进一步记录不同的颜色。通常采用色分离技术来获取不同的颜色，例如滤光片、3CCD和单CCD滤色镜等。滤光片是在线性CCD前面加装的光片，按照基色分为三等份，分别对应红色滤光片、绿色滤光片和蓝色滤光片。事实上，滤光片是用来选取所需辐射波段的光学器件，也就是只能让特定颜色成分的光线通过。这样在成像时通过滤光片的移动，使得CCD传感器能够分别记录红、绿、蓝相应基色下的光线强度，从而得到三基色所对应的三幅图像。然后，将三幅图像合成在一起，获得能显示的彩色图像。

3CCD则是使用了3个CCD进行成像，每一个CCD负责记录一种颜色。特别地，光线通过镜头时借助一个特殊设计的分光棱镜，将相应颜色的光线反射到相应的CCD。那么，每一个CCD产生一种颜色的图像数据，然后再进行合成。因此，经过一次扫描就可以得到彩色的图像。3CCD分色技术成像速度最快，但其造价最高，不适合于普通大众消费级相机使用。

单CCD滤色镜采用单个线性CCD，但在感光面上加入滤色镜，在感光的同时直接进行分色。由于原理简单、成本较低，单CCD滤色镜是目前数码相机广泛使用的色分离技术。

具体来讲，单CCD滤色镜大多采用彩色滤波阵列，也称为拜耳滤波器对入射光线进行滤波。该阵列上按照特定的模式分布着红、绿、蓝感光单元，也就是像素点。具体来讲，如图8.7(a)所示，每个绿点的四周，围绕着2个红点和2个蓝点。这样，拜耳滤波器实际上是一个4×4的阵列，包含8个绿色、4个蓝色和4个红色像素点。因此在整体分布上，绿点的数量是其他两种颜色像素点的两倍，能捕捉到的绿色分量也更充足。这也是基于人眼对绿色最敏感的视觉特性而做出的设计，使得记录的影像更符合人眼的观看效果。于是，光线经过镜头，被滤镜分解成一个个单色的光，并由传感器记录下每个点的光强数值，得到最原始的RAW数据，其文件的后缀名一般是.raw。这种数据只能呈现黑白马赛克，没有完整的颜色信息。为了获得彩色图像，需要进行去马赛克处理，将RAW数据还原为能反映真实世界中物体外观的彩色图像。

(a) 单 CCD 拜耳滤波器　　(b) 去马赛克

图 8.7　拜耳滤波器及去马赛克

常用的去马赛克处理主要包括邻近插值、线性函数插值和非线性核函数插值等方式。邻近插值是采用和当前像素点相邻的颜色分量，补齐该像素点缺少的其他分量。例如，无绿色的像素点，采用位于其左侧绿色像素点的强度；对于每一个红色和蓝色像素点，将其红、蓝强度值赋予右侧、下侧和右下角的像素点。这样，每个像素点都具有了红、绿、蓝的颜色分量，进而还原出彩色图像。

线性插值采用每个像素点的 4 邻域颜色分量进行插值，还原相应的图像像素颜色。例如，对于红色和蓝色像素点，其绿色分量是 $G(i,j)=\sum G(i+m,j+n)/4$，其中红色或蓝色像素点 (i,j) 的邻域对应 $(m,n)=\{(0,-1)(0,1)(-1,0)(1,0)\}$ 的像素集合。而对于蓝色像素点，其对应的红色分量是 $R(i,j)=\sum R(i+m,j+n)/4$，其中蓝色像素点 (i,j) 的邻域是 $(m,n)=\{(-1,-1)(-1,1)(1,-1)(1,1)\}$。对于绿色像素点，其对应的红色和蓝色分量分别是 $R(i,j)=\sum R(i+m,j+n)/2$ 和 $B(i,j)=\sum B(i+m,j+n)/2$，其中该像素点的邻域对应于 $(m,n)=\{(0,-1)(0,1)\}$ 或者 $(m,n)=\{(-1,0)(1,0)\}$ 的像素点集合。对于红色像素点，其对应的蓝色分量是 $B(i,j)=\sum B(i+m,j+n)/4$，其中红色像素点 (i,j) 的邻域是 $(m,n)=\{(-1,-1)(-1,1)(1,-1)(1,1)\}$。这样，对于红、绿、蓝的每个像素点，通过插值方式也都获得了缺少的颜色分量，从而能够还原出彩色图像。

从上面各个颜色分量的计算公式可以看出，线性函数插值方法计算简单，适合于在摄像机的 ISP 中实时处理。非线性插值则采用更复杂的非线性函数，例如双三次插值等进行红、绿、蓝颜色分量的插值。那么，经过插值后就得到了在显示器上能直接呈现的彩色图像。虽然非线性插值能够更好地还原颜色，但是计算较复杂。

虽然拜耳模式去马赛克提供了获得彩色图像最有效的一种方式，但是通过上述插值方式进行去马赛克后，生成的图像中会存在伪彩色、摩尔纹等问题。这里，伪彩色是由于拜耳滤色片上红、蓝、绿颜色点实际上是分布在不同的空间位置所造成的，从而导致去马赛克的插值过程中引入了拍摄场景中原本不包含的颜色。摩尔纹则是由差拍原理所产生的一种现象，也就是说如果感光元器件上像素的空间频率与影像中条纹的空间频率很接近，就会产生彩色的、高频率不规则的条纹，称为摩尔纹现象。为了解决插值去马赛克存在的这些问题，通常进一步在色度空间采用低通滤波等技术，对去马赛克后的图像做进一步的后期处理，通过后期校正的方式获得更加自然的彩色图像。

例题 8-1 拜耳成像的去马赛克处理。

问题：如下图所示的例子，写出使用线性插值进行拜耳成像去马赛克的伪代码。


解答：记彩色拜耳图像 I(左图)的宽和高分别是 m 和 n，输出的去马赛克图像为 D(右图)。那么，基于线性插值去马赛克处理的伪代码如下。

```
function demosaic (I, m, n)
    for i ← 0 to m-1 do
        for j ← 0 to n-1 do
            if (i%2 ==0 and j%2 ==0) then
                D[i][j][1] ← I [i][j]
                D[i][j][2] ← (I[i-1][j] +I[i+1][j] +I[i][j-1] +I[i][j+1]) / 4
                D[i][j][3] ← (I[i-1][j-1] +I[i+1][j-1] +I[i-1][j+1] +I[i+1][j
+1]) / 4
                else if (i%2 ==1 and j%2 ==1) then
                    D[i][j][1] ← (I[i-1][j-1] +I[i+1][j-1] +I[i-1][j+1] +I[i+
1][j+1]) / 4
                    D[i][j][2] ← (I[i-1][j] +I[i+1][j] +I[i][j-1] +I[i][j+1]) / 4
                    D[i][j][3] ← I [i][j]
                else
                    D[i][j][1] ← (I[i][j-1] +I[i][j+1]) / 2
                    D[i][j][2] ← I[i][j]
                    D[i][j][3] ← (I[i-1][j] +I[i+1][j]) / 2
                end if
            end if
        end for
    end for
end function
```

□

8.2.2 白平衡

白平衡，顾名思义，就是成像时白色的平衡。由于在有色光照射下，当我们看到白色时，白色可能会呈现出有色光的颜色，但我们仍认定它实际上应该是白色的。这主要是由于人的眼睛具有独特的适应性和自纠正性。但是，相机的传感器如 CCD 等，则不具有这种适应性。如果相机的色彩在调整时，和景物实际的照明色温不一致，就会发生偏色(如

图 8.8 所示)。许多人在使用数码相机拍摄照片时就会遇到这种问题：在日光灯下拍摄的室内照片显得发绿；而在白炽灯下拍摄的室内照片则显得发黄。白平衡就是根据实际拍摄场景的色温条件，通过图像处理，使得拍摄出来的图像能够抵消各种潜在的偏色。

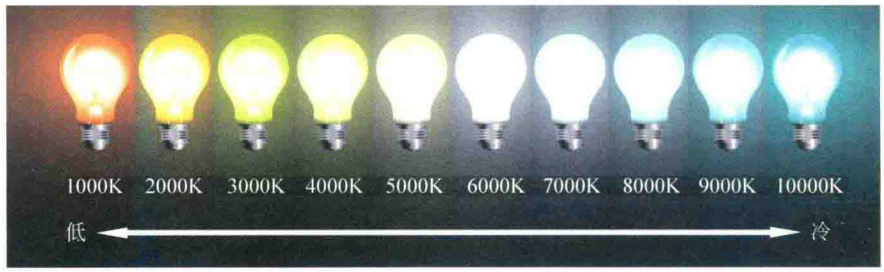


图 8.8 不同色温下的图像

色温，是以开尔文温度定量地表示色彩的一种色彩度量。英国著名物理学家开尔文认为，黑体能够将落在其上的所有热量吸收，而且没有损失；同时，又能够将热量产成的能量全部以光的形式释放出来。此时，人的眼睛所察觉到的颜色便会因受热力的高低而产生变化。白平衡的调整过程，其实就是通过调整色温来实现，分为手动白平衡和自动白平衡。

手动白平衡，是指人为选择真实光照环境下作为白色的参照物，也就是需要给相机指明场景中哪一个白色的物体确实是“白色”，将其作为白平衡的基准点。然后，通过分别调整 R、G、B 三个通道的颜色强度，使其能够表现出所挑选的白色参照物的颜色。这种方式往往需要用户具有比较丰富的摄影经验，才能应对不同环境下的白平衡调整，在照片中还原出拍摄场景中正确的色彩，使得照片的色彩更贴近真实。

自动白平衡，是基于灰色世界法则，设计相应的算法自动地对颜色进行调整。该法则假设图像中包含的反射面足够丰富，以至于可以作为自然界中所有景物的一个缩影。因此，如果假设这幅图片是在经典光源下拍摄的，那么各个颜色通道分量的平均值就应该等于灰色。这是基于灰色世界法则进行自动白平衡的基本原理。若这幅图是在非经典光源下拍摄的，那么均值就会大于或者小于灰色值。因此，自动白平衡就是通过调整每个像素的 R、G、B 值，使其均值等于灰色，进而实现自动白平衡。灰色世界法则中最重要的一点是“灰色”的定义和选择问题。例如，采用 R、G、B 各个通道最大值的一半，或者前面所述的平均值。

例如，采用颜色均值 K 作为预定义的“灰色”，也就是 $K=(R_{avg}+G_{avg}+B_{avg})/3$，其中 R_{avg}、G_{avg} 和 B_{avg} 分别是红色、绿色和蓝色分量的平均值。那么，对于每一个像素 (r,g,b)，在使用该灰色作为基准的白平衡处理后，红色分量变为 $r \cdot K/R_{avg}$，绿色分量变为 $g \cdot K/G_{avg}$，蓝色分量则变为 $b \cdot K/B_{avg}$。

除了灰色世界法则，自动白平衡方法还有镜面法，也就是假设图像中存在一个完全可以反射光源的镜面的点。那么，通过将该点调整为纯白色的颜色映射，就可以作为其余像素白平衡时的变换。但是，不论灰色世界法则，还是镜面法，都是根据一些经验假设而设计的自动方法。虽然它们在一定程度上可以减弱白色的偏差，但无法解决在任何光照环境下的白平衡问题，尤其是在复杂光照环境下。

例题 8-2 基于灰色世界法则的自动白平衡。

问题：如右图所示的输入图像(上图)，写出使用灰色世界法则进行图像自动白平衡(下图)的伪代码。

解答：彩色图像 I 的宽和高分别记为 m 和 n，输出的白平衡图像为 W。那么，采用颜色平均值的灰色世界法则来进行白平衡的伪代码如下。

```
function whiteBalance (I, m, n)
    b_avg ← 0
    g_avg ← 0
    r_avg ← 0
        for i ← 0 to m-1 do
            for j ← 0 to n-1 do
                r_avg ← r_avg +I[i][j].r
                g_avg ← g_avg +I[i][j].g
                b_avg ← b_avg +I[i][j].b
            end for
        end for
        r_avg ← r_avg / (m×n)
        g_avg ← g_avg / (m×n)
        b_avg ← b_avg / (m×n)
        k ← (r_avg +g_avg +b_avg) / 3
        for i ← 0 to m-1 do
            for j ← 0 to n-1 do
                W[i][j].r ← I[i][j].r × k / r_avg
                W[i][j].g ← I[i][j].g × k / g_avg
                W[i][j].b ← I[i][j].b × k / b_avg
            end for
    end for
end function
```

□

8.2.3 色调映射

色调映射是在能够对有限动态范围进行响应的成像传感器上近似地呈现高动态范围影像。色调映射的问题来源于 18 世纪的画家作画。画家在作画时所感受到的自然界中光线亮度变化的范围非常大，而使用的颜料颜色的范围却非常有限。因此，需要寻找一套行之有效的颜色转换方法，用有限的颜料画出光线亮度范围很宽泛的自然光。如今，这种颜色转换的方法就称为色调映射，也称为对比度修正。

如果简单地将真实世界的整个亮度域线性压缩到指定的狭小范围，就会在明暗两端同时丢失很多细节。色调映射就需要通过设计合理的映射方法来克服这一问题，使得位于动态范围两端的亮度在映射后也能正常显示。伽马校正是最常见的一种色调映射方法。这种方法利用指数函数曲线进行非线性色调编辑，在某种程度上能够检测出图像信号中的深色部分和浅色部分，并且自适应地使两者比例增大，从而提高图像对比度效果。

具体来讲，伽马校正的数学公式表示为

$$\boldsymbol{g} = \boldsymbol{f}^{\gamma} \tag{8.3}$$

其中，$\boldsymbol{f}$ 是原始图像，$\boldsymbol{g}$ 是校正后的图像，而不同 γ 值产生的色调映射效果也不同(如图 8.9 所示)。公式(8.3)形式简单，计算方便，因而被广泛用于色调映射。

从图 8.9 可以看到，当伽马值小于 1 时，在低灰度值区域内，动态范围变大，使得图像对比度增大；而在高灰度值区域内，动态范围变小。同时，图像整体的灰度值也变大。当伽马值大于 1 时，低灰度值区域的动态范围变小，高灰度值区域动态范围变大，降低了低灰度值区域图像对比度，提高了高灰度值区域图像对比度。同时，图像整体灰度变小。因此，在伽马校正时需要设置合适的 γ 值，以获得理想的色调映射结果。

(a) 原始图像　　(b) γ=0.45　　(c) γ=2.2

图 8.9　不同伽马值的色调映射结果

8.2.4　3A 调整

数码相机的 3A 技术包括自动对焦(AF)、自动曝光(AE)和自动白平衡(AWB)。这些技术全部采用自动的算法来调整拍摄时的相机参数，实现最佳的成像效果。其中，自动白平衡技术在前面已经讲述，接下来介绍另外两种自动调整技术。

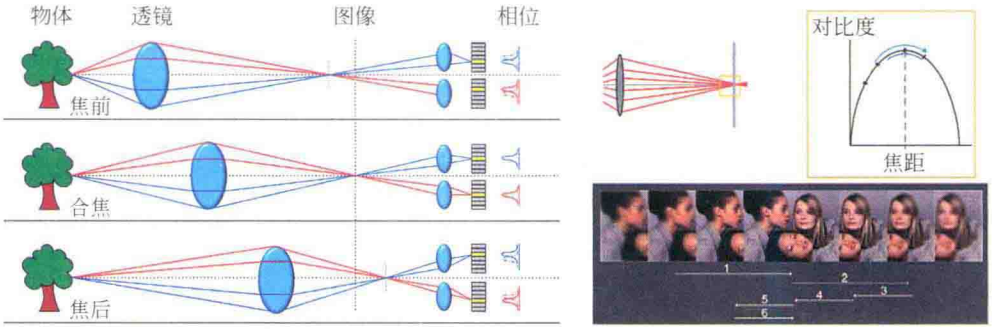
1. 自动对焦

自动对焦，是根据拍摄景物到镜头的距离将镜头调整到传感器前面的适当位置，以使得景物在传感器上能够清晰成像。自动对焦主要包括主动式和被动式两种方式。主动式采用 Time-of-Flight(ToF)方式，例如使用声波来测量相机和拍摄景物之间实际的距离，然后再调整镜头的焦距，以使得景物成像后的像点尽可能都落在成像平面上。实际使用时更多的是被动式，也就是不依赖于额外的硬件装置来估计景物到镜头的距离，包括相位检测、反差式、混合式等。

相位检测是利用两个位置不同的分离透镜，在传感器上同时形成两幅图像，然后检测出两幅图像之间的距离以及相应的两个波形。如图 8.10(a)所示，如果两个波形之间的距离恰巧为透镜焦距，说明拍摄的景物处于合焦状态，能够在成像平面上形成清晰的图像。如果距离大于焦距，那么就是焦前状态，反之就是焦后状态。这两种状态下都会产生模糊问题。这种方法可以直接计算出镜片需要移动的距离，以及镜片需要朝哪个方向移动，然后驱动镜头的移动以达到合焦状态。因此，相位检测的对焦速度十分迅速。只需要计算一次就可以完成对焦。然而，由于使用了分光技术，在弱光环境下很容易处于焦前或

焦后的对不上焦的状态。这就会使得拍摄出的画面感光不足、模糊严重。由于能耗较少，目前的单反相机大多使用相位检测对焦来完成自动对焦。

反差对焦则是分析镜头和传感器在不同距离下形成图像的对比度，通过寻找能使对比度最大的位置来确定焦距。如图 8.10(b)所示，只要通过测量得到的对比度在增加，镜头移动方向就会被保持，并按照预设的步长逐渐推进，直到合焦状态。反之，如果检测到低对比度，那么镜头就朝相反的方向逐渐运动，也就是直到合焦状态停止。反差对焦在弱光环境下也能正常工作，但是在处理器要计算较多的数据，能耗较高，而且往往导致对焦时间长，拍摄时会有一定的滞后。



(a) 相位检测　　(b) 反差对焦

图 8.10　自动对焦方式

混合对焦是结合上面两种对焦方式的优点，按照两步法进行对焦。首先，利用相位检测对焦速度快的特点来快速调整镜头和传感器之间的距离。然后，利用反差对焦进行微调。这也适合于弱光等环境。由于先将对焦位置进行了预先调整，在确定最大对比度时仅需要很少的时间也能够提供准确的对焦位置，这也就大大减少了对焦的时间。

2. 自动曝光

自动曝光，主要是自动调整光圈大小和曝光时间来控制镜头的光通量。通常采用测光方式来确定入射光线条件，从而自动地确定拍摄时所需要的合适的曝光量。常见测光方式有中央重点测光、局部测光和点测光三种。中央重点测光是假设画面中央部分的测光数据在整个画面中占绝大部分比例，而中央以外的测光数据作为小部分比例，以此确定拍摄时需要的曝光量。局部测光是对画面中预先指定的某一局部区域进行测光。点测光则是以位于中央极小范围内的区域作为曝光基准点，然后根据这个区域测得的光通量数据作为曝光依据。

8.3　计 算 摄 像

计算摄像属于计算摄像学的范畴，是一种通过光学编码和计算解码生成图像，并进行更高效的图像处理的过程。其实，数字摄像也可以看作通过透镜对光线传播进行编码的过程，借助透镜折射后的光线汇聚，来对景物表面上各点的颜色信息进行记录，也属于计算摄像的一种形式。但这里的计算摄像主要还是指结合光学技术、信号处理技术、数字图

像处理技术，进一步扩展传统摄像和数字摄像的功能，以及进一步提高数字摄像的质量。

计算摄像的成像过程不局限于小孔成像原理，通常使用新的光路编码获取信息，然后利用计算解码形成影像，也称之为计算成像(Computational Imaging)。换言之，计算摄像不再是借助透镜汇聚作用，简单地接收光线投影在传感器上形成的直接成像，而是经过编码再解码的方式进行成像。

8.3.1 计算成像编/解码

计算成像的编码和解码突破了以往成像方式对时间、空间和谱段的限制，实现了更丰富的成像功能。如图 8.11 所示，计算成像可以围绕空间、时间、深度、动态范围、视场、频谱等光的不同属性，通过对光路的编码和解码进行新形式的成像，实现成像质量的提升和成像功能的扩展。按照光路编码方式的不同，计算成像的编码方式可以笼统地分为物端编码、瞳面编码、焦面编码、照明编码、相机阵列编码、非传统成像编码等，而在解码时就需要借助相应的算法，通过计算方式来重构出目标影像。接下来简单介绍各种不同的编/解码方式及典型应用。

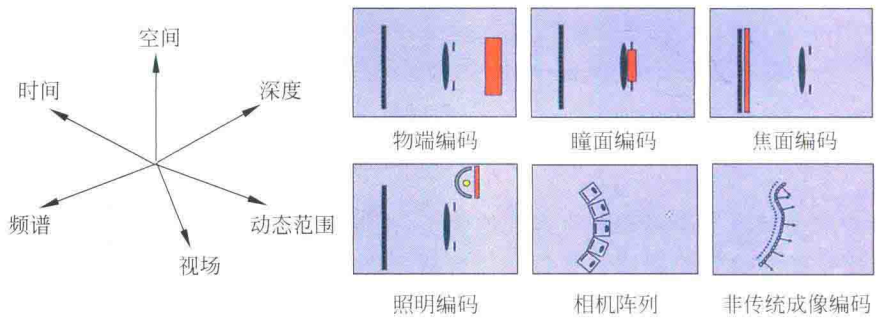


图 8.11 计算成像编/解码维度和方式(图片来自[96])

1. 物端编码

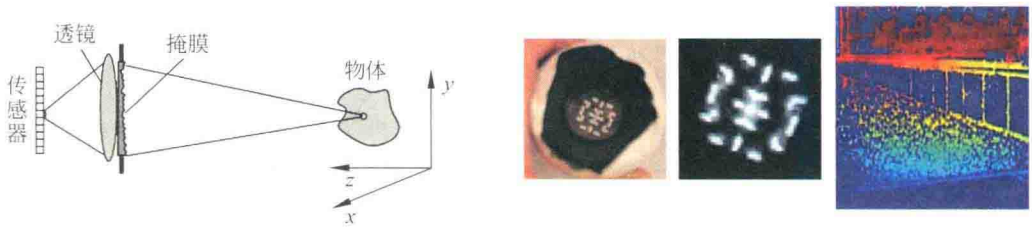
物端编码，是通过改变镜头前方和拍摄物体相关的光路进行计算成像。这是最容易实现的一种编码方式，大多数情况下不需要对镜头做任何的改动，而是借助已有的相机系统来实现计算成像。图 8.12 展示了三种物端编码的案例。其中，图 8.12(a)是通过在镜头前放置全反射的半球镜面或者抛物镜面，把普通相机变为全景相机。这种镜面的使用可以将更大视角范围的光线进行汇聚，然后再投射到相机的传感器上，也就是借助镜面对光线的汇聚作用对入射光线进行更大视场的编码。这样就能够实现更大视角范围的全景成像。图 8.12(b)是在镜头前放置雾面玻璃，利用不同远近物体成像后的模糊程度差异，估计所拍摄物体的深度信息。该方法实际上利用了雾面玻璃的透光特性，对进入相机的光线进行编码。距离镜头不同远近的物体反射光线经过雾面玻璃后的汇聚效果有所差别，形成不同程度的散焦模糊。那么，通过去模糊进行计算解码，就可以一方面获得清晰的图像，另一方面也获得了所拍摄物体的深度信息。图 8.12(c)是在镜头前放置三棱镜，通过分光编码，捕捉不同波长范围的光线强度信息，进而实现多光谱视频的拍摄。

(a) 全景相机　　(b) 深度复原　　(c) 多光谱相机

图 8.12 物端编码的计算成像(图片来自[97-99])

2. 瞳面编码

瞳面编码是通过改变出入透镜瞳面的光路进行计算成像。图 8.13 展示了两种瞳面编码的案例。其中,图 8.13(a)是一种基于光圈相位编码的超分辨率方案。该方案借助伪随机编码的光圈,产生亚像素级别的成像。然后,利用不同相位差下成像结果的叠加,重构出高分辨率的影像。图 8.13(b)是通过设计特定模式的光圈,使得成像后的影像在经过傅里叶变换后,能够有效地通过零值分布来反求散焦模糊核函数,克服了传统圆形光圈求解模糊核时的病态问题。在此基础上,根据模糊核尺寸和物距的关系,能够进一步恢复所拍摄场景的深度信息。



(a) 光圈相位编码的超分辨率　　(b) 深度复原

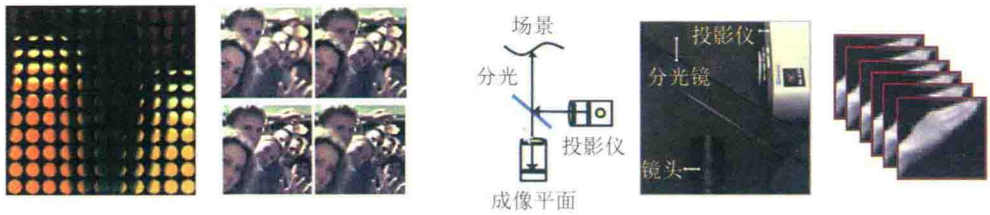
图 8.13 瞳面编码的计算成像(图片来自[100-101])

3. 焦面编码

焦面编码,是通过改变位于焦平面位置的传感器的入射光路进行计算成像。图 8.14 展示了两种焦面编码的案例。其中,图 8.14(a)是在焦平面放置微透镜组,对透过镜头的光线按照入射角度进行空间编码。那么,在成像时就可以根据对焦位置的要求,自动地挑选相应角度的光线强度解码,重构出符合对焦位置的图像。图 8.14(b)是一种通过焦面编码实现的高速摄像系统。该系统借助投影仪来实现逐像素的曝光时间控制,使得在焦平面成像时能够根据场景内容,在时间和空间维度自适应地进行非均匀采样。然后,通过对不同曝光时间下的像素强度进行线性规划来优化处理,重构出高分辨率、高帧率的视频帧序列。

4. 照明编码

照明编码是通过改变相机闪光灯的照明状态对入射光路的光线强度分布进行编码,然后再通过解码进行计算成像。其中,图 8.15 展示了两种照明编码的案例。图 8.15(a)是利用多路 LED 灯产生指定模式的照明,从而生成包含多种谱段的光照效果。然后,采



(a) 微透镜阵列的重对焦　　(b) 高速摄像

图 8.14　焦面编码的计算成像(图片来自[102-103])

用普通的 RGB 相机记录物体表面的反射强度,通过重构来获得更大波长范围的光谱信息。图 8.15(b)则是通过有规律地改变光照实现对拍摄物体三维模型的重建。该方法利用投影仪投射指定模式的结构光,通过相机捕捉的物体表面信息重构其三维形状。

(a) 多路 LED 照明的多光谱成像　　(b) 变光照三维重建

图 8.15　照明编码的计算成像(图片来自[104-105])

5. 相机阵列编码

相机阵列是通过使用一组相机阵列来提高入射光路的视角分布,从而进行计算成像。这种方式主要是为了能够在更大视场范围内拍摄场景。图 8.16 展示了一种相机阵列的案例。这里,采用一组在空间不同位置排布的相机来同时拍摄不同视角的视频(如图 8.16(a)所示),这样就能够对拍摄场景进行全方位的覆盖。为了能够产生更大视场范围,在排布相机时使得视角朝向差异尽可能大。然后,对每一个相机所拍摄的视频,按照第 7 章中介绍的视频拼接方法进行组合,就可以获得如图 8.16(b)所示的动态全景视频帧序列。

(a) 相机阵列　　(b) 全景视频

图 8.16　相机阵列的计算成像(图片来自[106])

6. 非传统成像编码

非传统成像编码是在传统相机的基础上使用更加复杂的材料、装置等进行新型的计

算成像。例如，使用可卷曲的传感器进行任意形状物体的拍摄和成像，以及使用特殊的微波材料进行感知成像等。这类计算成像方式能够获得以往相机难以直接拍摄的成像效果，大大拓展了传统成像和数字成像的成像功能。

8.3.2 性能分析

计算成像扩展了传统相机和数码相机的成像功能。然而，计算成像需要首先解决一个基本问题，这就是计算成像是否真正地提高了成像效果，也就是说信息量是否通过计算方式产生了增益。这就涉及对计算成像性能的分析，也就是如何度量计算成像效果。

信噪比(Signal-to-noise ratio，SNR)，是数字系统中经常使用的性能评价指标，定义为系统中信号与噪声的比例。信号，通常是指来自外部需要通过设备进行处理的电子信号。噪声，是指经过该设备处理后产生的原信号中并不存在的无规则的额外信号，并且不随原信号的变化而变化。计算成像在编码过程中，通常需要将采集的光通量增大，从而提高了 SNR。而在解码计算时，噪声也会随之产生，使得 SNR 值反而减小。那么，计算成像效果就取决于选择合适的编/解码方案，使得两者之间达到合理的平衡，将成像效果做最优化处理。

为了定量分析计算成像的性能，通常选择计算成像和传统成像都能处理的问题，例如去散焦模糊、去运动模糊等问题，对处理结果的信噪比做量化，进行定量的分析和评判。这类问题在数学上都可以表示为如下形式的成像模型：

$$g = \boldsymbol{H} \cdot f + \eta \tag{8.4}$$

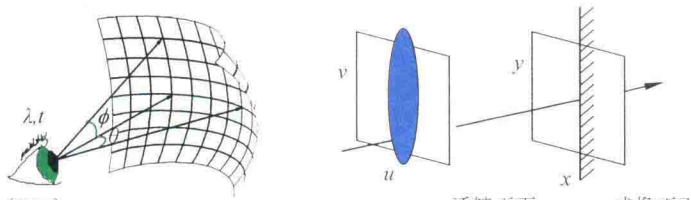
其中，g 是观测数据，也就是实际成像所形成的图像，f 是原始数据，$\boldsymbol{H}$ 是编码掩膜对应的观测矩阵，η 则是噪音。对应于普通相机的传统成像模型，公式(8.4)中的观测矩阵 $\boldsymbol{H}$ 是单位矩阵，也就是原始数据加上一定程度的噪音后形成普通相机拍摄后的图像。在公式(8.4)的基础上，经过大量实验分析，在小于 125 流明光照条件下，计算成像会比以往的成像方式产生更高的信噪比，也就是说确实是能够提升传统成像和数字成像的效果；反之，计算成像不会对最终的成像效果带来实质性的提升。

8.4 计算光场成像

光场，顾名思义，就是描述光的某些物理量在空间中的场分布。这个概念第一次被明确提出是在 1939 年，俄罗斯物理学家 Arun Gershun 在他的一篇论文中使用了光场，后来被 MIT 教授 Edward Adelson 和 James Bergen 在 20 世纪末的一篇计算机的论文中加以完善，并给出了全光函数 (Plenoptic Function)的数学形式。具体来讲，光场描述了空间中任意一点朝任意方向投射光线的强度。如图 8.17(a)所示，完整描述光场的全光函数 LF 是具有如下形式的七维函数：

$$LF(x, y, z, \theta, \varphi, \lambda, t) \tag{8.5}$$

这里，该函数的参数包含了任意一点的空间位置(坐标分量分别是 x、y、z)、方向(极坐标分量为 θ、φ)、波长(λ)和时间(t)。事实上，波长 λ 在一定程度上决定了所投射光线的强度。



(a) 七维全光函数　　(b) 四维光场函数

图 8.17　光场模型

在实际成像系统中，波长和时间维度的信息通常是用 RGB 通道和不同时刻对应的帧图像来表示。因此就光场而言，只需要关注光线的方向和位置就可以了。这样，全光函数就从七维降到了五维。此外，大部分成像系统中的光线都是通过光圈和透镜系统被限定在一个有限范围的光路里进行传播，所以光场就可以使用更简单的两个平面来表示，从而将公式(8.5)所示的七维光场函数进一步简化为四维光场函数 $LF(u,v,x,y)$。如图 8.17(b)所示，这两个平面则分别对应了相机的镜头平面(u 和 v)和成像平面(x 和 y)。计算光场成像主要就是围绕这两个平面的光路编码和解码进行处理，进而扩展以往成像的效果。接下来，介绍两类典型的计算光场成像技术，包括基于光场的重对焦成像技术和光圈编码成像技术。

8.4.1　基于光场的重对焦成像

普通相机在拍摄瞬间只会捕捉在一个光面上进行对焦时所形成的图像。这样，位于对焦平面上的物体成像清晰，而其余部分则会发生焦外模糊。这是由于经过普通相机透镜的汇聚，物体上来自同一点发射的不同方向的光线都汇聚成图像上的一个点。因此，获得的图像只记录了光线强度的信息，而丢失了光线来自于哪个方向的信息。光场则是包含了各个光线方向的信息。利用光场成像，就可以记录下所有方向投射光线的数据。这样，通过后期在计算机中重新选择对焦位置，就可以获得重对焦的图像。基于这种原理构造的相机也称为光场相机。

以 Lytro 为代表的光场相机为例，它通过将焦平面改为微透镜的阵列来同时记录光线的方向信息，也就是对更多投射光线的方向进行了编码。如图 8.18 所示，这些微透镜将汇聚到焦平面上的光线进一步发散到不同方向，从而能够获取物体上同一点发射的不同方向的光线信息。因此，光场成像是对不同方向的光线进行编码，然后记录光线强度和光线方向。这里就是使用四维的光场函数 $LF(u,v,x,y)$ 来表示记录的光线信息，实际上表示了入射光线的不同方向。而在解码成像时，通过指定焦点位置来选择一个成像平面，将光场函数重新参数化到这个平面上，然后把对应于同一点的光线进行积分，就可以得到和该成像平面相对应的光线强度，进而形成符合该焦点位置的新图像。

光场相机最大的优势之一就是可以利用光场编码和计算成像，实现了先拍照后对焦这一传统相机所不具备的成像功能，一定程度上方便了用户拍照。

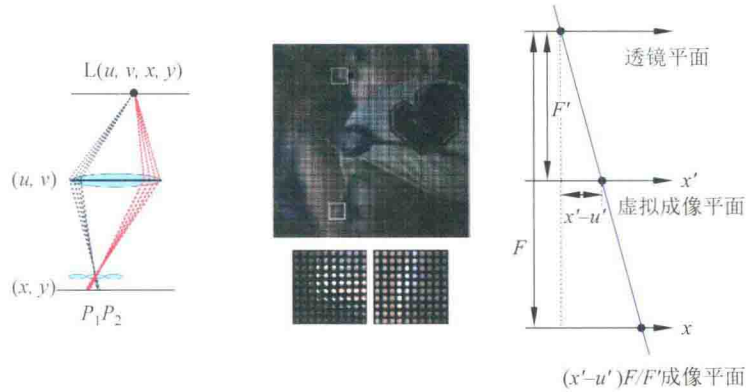


图 8.18 基于光场的重对焦成像(图片来自[102])

8.4.2 基于光圈编码的去散焦成像

常见相机镜头大多使用圆形光圈,不可避免地产生散焦模糊。所谓散焦模糊,是指景深以外物体的成像点没有落在焦平面上,从而形成具有一定面积的成像区域,造成物体表面信息的混叠。在数学上,散焦模糊的形成可以看作是清晰图像和散焦模糊核进行卷积操作的结果,而散焦模糊核对应于点扩散函数(Point Spread Function,PSF)。不同程度的散焦模糊对应了不同尺寸和形状的模糊核,也就是不同物体成像在同一平面上的弥散圆的大小。

传统相机所使用的圆形光圈对应的散焦模糊核也是圆形的。然而,圆形光圈存在以下问题:卷积过程会削弱高频信息,而且核函数在频域上会有多个零值点。光圈编码则是采用其他模式的光圈设计方式,将圆形光圈变成不规则形状的光圈。这种形状的不规则性就会影响核函数的取值分布,可以有效减少核函数零值点的数目,从而最大限度地消除不同尺度的光圈的影响。

具体来讲,因为傅里叶变换后的频域零值点区间大小决定了模糊核函数的尺寸,所以相比于圆形光圈,编码光圈的零值点分布更为分散(如图 8.19 所示)。这样就可以在更大尺度范围内避免零值点,从而更容易地选择合适的尺度进行反卷积。在得到模糊核函数后,再借助常用的去模糊方法,例如最大后验概率优化,对模糊图像进行反卷积,就可以得到去模糊后的图像。

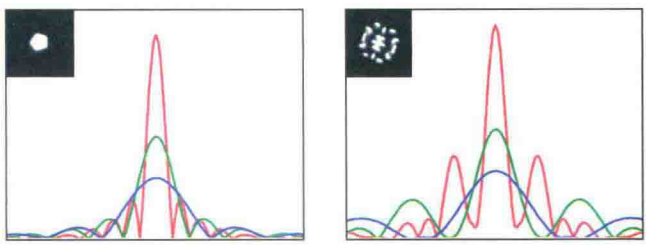


图 8.19 圆形光圈和编码光圈的核函数分布(图片来自[101])

8.5 计算光谱成像

光谱，又称光学频谱，是复色光的光线经过色散系统（如棱镜、光栅）分光后，散开的单色光按波长（或频率）大小依次排列所形成的图案（如图 8.20 所示）。按照光线波长分布，可以将光谱分为紫外光谱、可见光谱、红外光谱等。紫外光谱的波长范围是 1nm 到 380nm。可见光谱的波长范围是 380nm 到 780nm，是人眼能够感知的光谱范围。红外光谱的范围则是超过 780nm 的区间，而根据波长大小又可以分为近红外区、中红外区和远红外区。

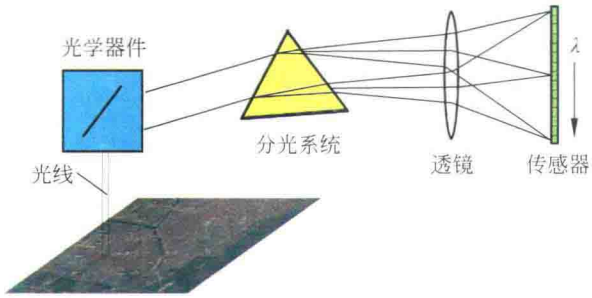


图 8.20 光谱色散

按照光谱的产生方式，可以分为发射光谱、吸收光谱、散射光谱等。发射光谱，是物体自行发光形成的光谱。吸收光谱，是连续光谱被吸收部分波长的光线后形成的光谱。散射光谱，则是光线照射到物质上形成的非弹性散射光谱。光谱成像的目的就是为了获取这些不同波长的光的强度。事实上，光谱信息被称为物质的“基因”，反映了物质成分的本质特征。因此，光谱成像在航天遥感、环境监测、安检安防等有着重要应用。

8.5.1 传统光谱成像

遥感是一种最典型的光谱成像途径。遥感传感器接收的光谱会受到大气过滤（散射、吸收）等影响。目前，遥感系统根据光谱来源主要分为三类：太阳辐射散射、热红外和雷达波。太阳辐射散射，主要用于检测地球表面不同物质对太阳辐射进行散射后的光谱分布，这样就能够根据不同波长来区分物质的属性差别。因为需要依靠太阳光源，这种方式只能在白天获取相应的光谱信息，而且受大气状况影响较大。热红外，是地表物质吸收太阳光后发射的光谱，反映了物质的热属性。热红外对于白天和晚上的温差比较敏感。太阳辐射散热和热红外属于被动遥感方式。雷达波成像则是主动式，通过发射雷达波和检测接收到的返回雷达波的变化，判别物质的属性及成分。由于具有比较强的穿透力，雷达波成像主要用于云层较厚情况下的对地观测。

在遥感成像时，通常利用光谱反射率来量化不同物质的光谱属性。反射率定义为以波长为变量、反映接收到的反射能量和入射能量的比率函数。以石油为例，在可见光和近红外，反射率和波长是呈线性递增，在中红外达到顶峰。而对于植被，在可见光范围内反

射率较低，但在近红外的边界处有一个大的跃升。此外，清澈的水体则基本能够吸收所有大于可见光波长的光强。因此，不同的物质所呈现出的光谱响应具有明显的差异，可被用于物质的探测。

以遥感成像为代表的传统光谱成像在使用各种光谱仪采集光谱数据时，在空间分辨率、谱分辨率和辐射分辨率等方面还存在一些瓶颈问题。空间分辨率，是图像细节的一种度量，通常表示为像素单元覆盖的地面尺度，反映了成像的清晰度。空间分辨率和传感器的设计以及拍摄距离和环境等有密切关系。谱分辨率，是指可区分的波长粒度，分为多光谱、高光谱、超光谱等。辐射分辨率，是指用于量化光谱强度的位数，也就是采用多少位的二进制来表示采集的光谱强度。使用的位数越多，能够区分的波长也就越精细。

传统光谱成像在解决上述空间、谱段等的分辨率问题时，一种方式是通过提高传感器性能来实现，从物理上提高传感器能够接收光谱的灵敏度。但是这种方式和传感器的物理特性有着极大的关系。另一种方式则是通过计算的方式，通过光路编码和解码过程，提高空间、谱的分辨率，称为计算光谱成像。下面介绍几种计算光谱成像方法。

8.5.2 基于掩膜分光镜的计算光谱成像

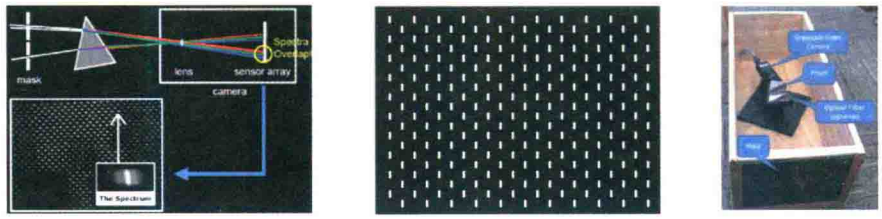
计算光谱成像，是在传统色散型光谱成像技术的基础上，通过在光路中引入适当的编码模板来对采集的数据进行调制。然后，通过图像处理手段进行解码，实现空间与光谱信息的高效成像，进而解决传统光谱成像技术光通量低、逐行扫描成像时间长、分辨率低等缺点。通过光谱成像可以获取目标场景的连续光谱采样图像，其中包含了更为丰富的光谱信息和空间信息。这里，采用掩膜分光镜是一种典型的计算光谱成像方法。

普通相机拍摄彩色图像时，成像介质 CCD 通常记录 RGB 三个通道或者说三个波段的光强信息。为了记录更多波段的光强信息，可以采用三棱镜对入射光线进行分光处理，主动地将不同波段的光谱进行分离，从而能够获取更高分辨率的光谱数据。物理上，三棱镜能够将波长不同的光线，按照不同的角度进行折射（例如波长越大，折射率越小），这样就可以达到分光的目的。然而，通过相机直接记录三棱镜分光后的光线，会产生频谱重叠的现象。这是由相近光线中相同波长的光谱覆盖区域重叠所造成，从而不能很好地区分不同波长的光强信息。

这里，如图 8.21(a)所示，可以将三棱镜放置在一个掩膜后面，通过掩膜的空间调制来解决上述频谱重叠问题。掩膜是由一些均匀分布的小孔组成的，而不同小孔之间存在一定范围的遮挡（如图 8.21(b)所示）。这些小孔实际上就对入射光线进行了空间的调制，使得穿过掩膜的光线能够彼此在空间上形成间隔，也就是不同波长的光谱能够彻底分离开来。那么，再通过三棱镜分光后，就可以在成像平面上避免相邻谱段的重叠。

一般而言，通过掩膜孔洞的光线经过三棱镜分光后，形成的频谱宽度和透镜焦距、波长、孔洞尺寸等因素都有直接关系。例如，焦距增加，频谱宽度增加；而孔洞之间的距离也影响频谱的空间分布。图 8.21(c)展示了一个采用单色（灰度）相机记录相应分光频谱光强的装置。这里，放置在三棱镜前的掩膜孔洞的大小为 0.2mm×1mm，邻近孔洞的距离 5mm。这种配置可以将不同谱段的光线做合理的分离，达到光谱成像目的。

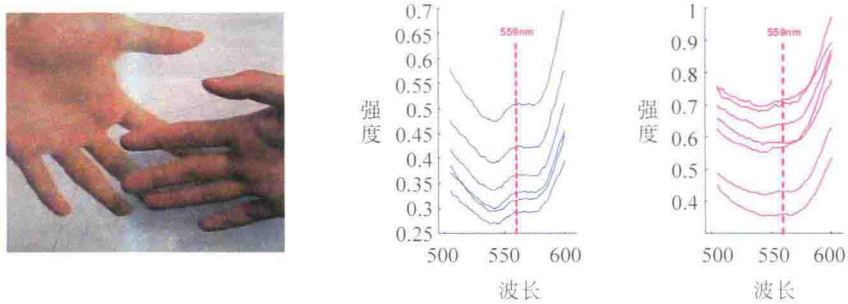
利用计算光谱成像获得的多光谱信息，可对物体材质进行更准确检测和识别。事实



(a) 成像原理　　(b) 掩膜　　(c) 装置

图 8.21　基于掩膜分光镜的多光谱成像系统(图片来自[99])

上，对于光谱图像，沿着光谱维采样所形成的向量称为光谱曲线。如图 8.22(b)所示的光谱曲线，人的皮肤在波长为 559nm 处会出现一个 W 形极值。这样，通过该极值是否存在就可以区分真实的皮肤和假的皮肤，进而能够对图 8.22(a)中所展示的真手和假手进行判别。这里假手是通过彩色打印在纸张上的手的图像，其颜色值和真手很相像，但材质却不是皮肤。正是因为能够将 559nm 附近的光谱区分开来，才使得 w 型极值在光谱图像中出现，从而可以利用该特性对真手和假手进行判别。这也充分说明了计算光谱成像对提高光谱信息利用的重要作用。



(a) 真手和假手　　(b) 光谱

图 8.22　基于多光谱的真假皮肤检测(图片来自[99])

8.5.3　基于双相机压缩感知的计算光谱成像

多光谱、高光谱、超光谱等光谱图像本质上是三维体数据，包含二维空间维和一维光谱维。这也使得在谱段数目增加时，获取高分辨率光谱图像的难度不断加大，尤其是对于高光谱和超光谱图像。为此，美国 Duke 大学的 David Brady 教授利用压缩感知提出了编码孔径快照式光谱成像系统(Coded Aperture Snapshot Spectral Imaging，CASSI)，其结构原理如图 8.23 中的 CASSI 支路所示。

CASSI 系统利用编码孔径和色散棱镜等光学器件，对场景的光谱信息进行空间编码以及空间调制，实现了对三维光谱图像的二维压缩光谱采样，并使用压缩感知对原始三维高光谱图像进行计算重构。该系统具有成像速度快、能获取动态光谱信息的特点，但是空间分辨率不高。

为了进一步提高光谱图像的空间分辨率，提出了如图 8.23 所示的压缩光谱成像系统

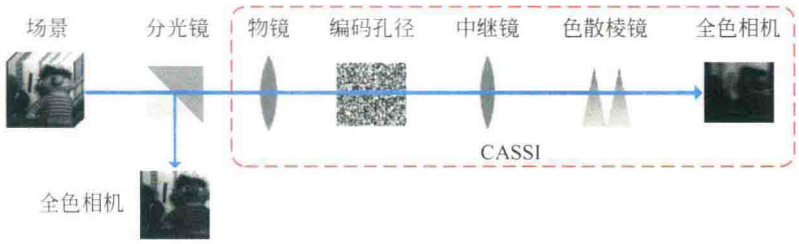


图 8.23 双相机压缩光谱成像系统(图片来自[109])

(Dual-camera Compressive Hyperspectral Imager, DCCHI)。与 CASSI 系统相比,DCCHI 系统的光谱图像重建的质量更高,能够完成高分辨率高光谱图像和高光谱视频数据的获取。具体来讲,来自场景的入射光到达 DCCHI 系统以后,首先经过分光镜被一分为二:一半进入 CASSI 系统,另一半则直接到达全色相机。其中,进入 CASSI 分支的光线经过物镜到达编码模板,编码模板会对光谱信息进行空间编码,然后由色散棱镜在竖直方向上进行色散,最后到达全色相机,获得二维混叠采样。而通过分光镜产生的另一路光线则直接进入另一个全色相机。最终,通过结合两个全色相机获得的图像就可以重构高分辨率的光谱图像。

如果用 $F(m,n,\lambda)$ 表示高光谱图像,其中 $1\leqslant m\leqslant M$ 和 $1\leqslant n\leqslant N$ 分别是图像空间的行、列索引,$1\leqslant\lambda\leqslant\Lambda$ 是谱段索引,那么图 8.23 中 CASSI 分支在全色相机上进行成像的数学模型可以写为

$$g_c(m,n)=0.5\sum_{\lambda=1}^{\Lambda}\omega(\lambda)\varphi(m-\lambda,n)F(m-\lambda,n,\lambda) \tag{8.6}$$

其中,观测值 $g_c(m,n)$ 表示像素 (m,n) 的光强,$\varphi(m,n)$ 为编码孔径处的二值编码模板,而 $\omega(\lambda)$ 表示该全色相机的光谱响应。公式(8.6)可以写成如下矩阵形式:

$$\boldsymbol{G}_c=\boldsymbol{H}_c\cdot F \tag{8.7}$$

其中,$\boldsymbol{H}_c$ 就表示了 CASSI 系统的前向响应矩阵,包含了 $\omega(\lambda)$、$\varphi(m,n)$、色散以及积分的共同作用来形成混叠图像 $\boldsymbol{G}_c$,同时该矩阵大小为 $M(N+\Lambda-1)\times MN\Lambda$。

直接进入全色相机分支的光线则会直接被全色相机成像,形成二维压缩未混叠采样。该过程的成像模型是

$$g_u(m,n)=0.5\sum_{\lambda=1}^{\Lambda}\omega(\lambda)F(m,n,\lambda) \tag{8.8}$$

其中,$g_u(m,n)$ 就表示了全色相机分支在位置 (m,n) 的观测值。公式(8.8)可以写成如下矩阵形式:

$$\boldsymbol{G}_u=\boldsymbol{H}_u\cdot F \tag{8.9}$$

其中,$\boldsymbol{H}_u$ 表示全色相机的前向响应矩阵,包含 $\omega(\lambda)$ 和积分的共同作用来形成未混叠图像 $\boldsymbol{G}_u$,同时该矩阵大小为 $MN\times MN\Lambda$。

在此基础上,将公式(8.7)和(8.9)联立,就得到了整个 DCCHI 系统的成像模型:

$$\boldsymbol{G}=\boldsymbol{H}\cdot F \tag{8.10}$$

其中,观测矩阵 $\boldsymbol{G}=[\boldsymbol{G}_c;\boldsymbol{G}_u]$,大小为 $M(N+\Lambda-1)+MN$;前向响应矩阵 $\boldsymbol{H}=[\boldsymbol{H}_c;\boldsymbol{H}_u]$,

大小为$[M(N+\Lambda-1)+MN]\times MN\Lambda$。那么，待重建的高光谱图像 F 就是要从观测矩阵 $\boldsymbol{G}$ 中计算获得，这也是计算光谱成像系统需要解决的重要问题。

事实上，观测 $\boldsymbol{G}$ 的数据量远小于高光谱图像 F 的数据量，因而这是一个欠定问题。根据压缩感知原理，F 可以通过求解如下一个带约束的目标函数得到：

$$\hat{F}=\underset{F}{\operatorname{argmin}}\frac{1}{2}\|\boldsymbol{G}-\boldsymbol{H}\cdot F\|_2^2+\tau\|\boldsymbol{D}F\|_1 \tag{8.11}$$

其中第一项是数据保真项，第二项为正则化约束项，此处使用的是全变差约束项，$\boldsymbol{D}$ 为差分矩阵。通过使用交替方向乘子法对公式(8.11)进行求解，就可以完成对高光谱图像 F 的计算重建。这里需要注意，正是因为使用了额外的全色相机直接获取高分辨率全色图像，所以大大提高了高光谱图像的空间分辨率。

8.5.4 基于成像机理的计算光谱复原

光谱图像记录了不同波长的光线强度信息。然而，正如 8.1.2 节中介绍的像差，由于透镜对不同波长光线的折射率的差异，使得不同谱段的光谱图像不可避免地呈现出不同程度的散焦模糊。虽然可以通过多个透镜构成的镜头组来抵消不同波长光线成像时的像差，但是这会大大增加镜头设计的复杂度和体积，尤其是当采集高光谱(几十到几百个谱段)、超光谱(几百到几千个谱段)等具有更多谱段的光谱图像时，很难在所有谱段都能使光谱清晰成像。因此，如何将光谱图像中出现散焦的谱段对应的模糊图像通过计算的方式进行复原，也就是去除这些图像上的散焦模糊，得到清晰的光谱图像，是计算光谱成像中的一个重要问题。

从图 8.20 所示的光谱成像的机理可以看出，当成像元器件所在的平面位置固定时，某些波长对应谱段的光线能够在成像平面上合焦，获得清晰的光谱图像。这些波长对应的谱段一般称为基准谱段。那么，大于或小于该基准谱段波长的光线就无法在成像平面合焦，产生散焦模糊。如图 8.24 所示，在包含 16 个谱段的光谱图像中，第 9 个谱段对应于基准谱段，可以看出其具有清晰的成像。其余谱段则或多或少存在着模糊现象，尤其是远离基准谱段的那些图像，例如第 1 个谱段和第 16 个谱段，在边缘处具有很明显的模糊。因此，可以利用基准谱段的光谱图像作为参考，对其余谱段的散焦图像进行复原。

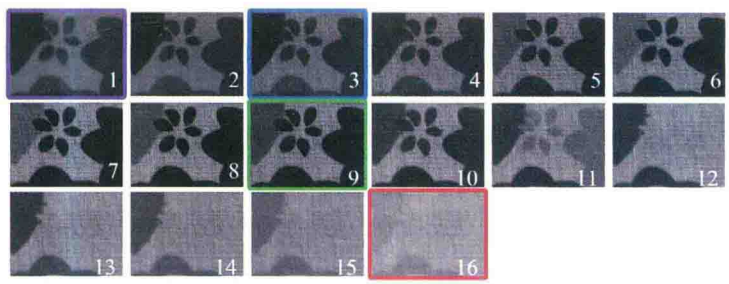


图 8.24 包含 16 个谱段的光谱图像

然而，由于不同谱段实际上是记录了拍摄物体不同的物质成分，在光谱图像上还是会

体现出一定的内容差异。仅仅依靠散焦图像和基准谱段图像的内容相似性，还是很难进行准确复原。例如在图 8.24 中，在第 13 个谱段之后的光谱图像中已经很难看到前面谱段的图像中出现的花瓣图案。因此，除了基准谱段的信息，还需要从散焦光谱图像的成像机理对其他谱段的光谱图像进行处理，才能获得更准确的复原图像。

事实上，散焦模糊和成像平面上形成的弥散圆紧密相关。弥散圆是离焦光线形成的圆斑，区别于合焦光线形成的点。当超过容许弥散圆半径时，就会产生相应的模糊。因此，可以利用弥散圆和散焦模糊的关系，通过更准确的模糊核估计来提高散焦光谱图像复原时的质量。

具体来讲，薄透镜成像时遵循透镜制造者公式(Lensmaker's Formula)，它描述了波长为 λ_i 的光线经透镜成像时的焦距 f_i 和折射率 n_i 之间应当满足的关系：

$$\frac{1}{f_i}=(n_i-1)\left(\frac{1}{C_1}-\frac{1}{C_2}\right) \tag{8.12}$$

其中，C_1 和 C_2 是构成薄透镜的两个球面的曲率。此外，物距为 d 的物体形成的弥散圆半径 r_i 可以写为

$$r_i=\frac{\mathrm{d}A}{f_i}\frac{|f_i-\tilde{f}|}{d-\tilde{f}} \tag{8.13}$$

其中，A 是光圈的半径，$\tilde{f}$ 对应于基准谱段的光线经透镜成像的焦距，也就是成像平面到透镜的距离，如图 8.25 所示。此外，公式(8.12)和公式(8.13)也反映了弥散圆的半径和折射率之间的直接关系。

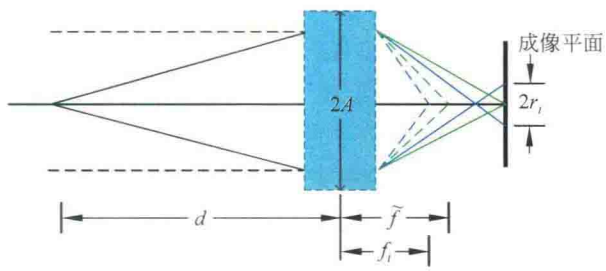


图 8.25 弥散圆产生的散焦模糊

通常情况下，散焦模糊的数学模型可以描述为高斯点扩散函数 $G(\sigma)$，其中 σ 是标准方差。那么，根据光线波长 λ_i 和折射率 n_i 之间的关系(物理上称为柯西色散公式)，就可以进一步得到波长 λ_i 及其引起的模糊核函数标准方差 σ_i 之间如下的近似关系：

$$\sigma_i\approx\frac{a_i}{\lambda_i^2}+\frac{b_i}{\lambda_i}+c_i \tag{8.14}$$

该公式是关于 $1/\lambda_i$ 的二次多项式，而其中的未知量是该多项式的系数 a_i、b_i 和 c_i。对于高光谱或者超光谱图像，能够通过不同谱段的最小二乘拟合来求解这些二次项系数，进而估计散焦模糊核函数。在此基础上，再借助常用的去模糊方法，例如最大后验概率优化，对散焦谱段的图像进行反卷积操作，就可以得到去模糊后的光谱图像。

8.6 小　　结

受光学系统、成像机理等因素的影响，传统成像、数字成像后的影像处理往往在功能和质量方面的效果有限。计算摄像则是在空间、时间、深度、动态范围、视场、频谱等维度进行成像的功能扩展和质量提升，打破了以往影像处理的局限，为信息的获取、分析和利用提供了新的途径。

思考题

8.1　摄像学的发展经历了哪些阶段？各个阶段的特点分别是什么？
8.2　小孔成像的基本原理是什么？孔径大小如何影响成像的清晰程度？
8.3　透镜成像的基本原理是什么？单一透镜成像存在哪些问题？
8.4　白平衡的问题是怎样产生的？如何进行自动白平衡？
8.5　数字成像时的 3A 调整包括哪些操作？各自如何进行自动调整？
8.6　计算成像的定义是什么？它与传统成像、数字成像的区别有哪些？
8.7　计算成像时光信息的编/解码方式包含哪些？各自对应的典型应用有哪些？
8.8　光场的定义是什么？常用的数学模型是什么？
8.9　计算光谱成像可以解决传统光谱成像的哪些问题？

第9章 计算机动画

动画(Animation)是运动(活动)的画面(图像)。动画是通过连续画面的播放,给人的视觉造成随时间动态变化的效果。动画涵盖广泛的内容,包括影视动画片、影视特技动画、广告动画、游戏动画等,在日常生产生活中被大量使用。计算机动画就是由计算机程序生成的动画,又可以分为二维动画和三维动画。其中,二维动画是通过设计和绘制二维图形或图像而生成连续的画面;三维动画,则是通过几何建模和绘制,以及直接或间接地控制三维模型运动来生成连续的画面。计算机动画给人们提供了一种充分展示想象力和艺术创作的新方式。

本章介绍计算机动画的基本原理和制作方法,重点介绍关键帧插值、运动捕捉、物理模拟等常用的计算机动画生成方法。此外,简单介绍群体动画的制作方式。

9.1 动画制作

动画是运动中的艺术。正如匈牙利动画大师 John Halas 所讲的,运动是动画的基本要素。通过人眼视觉所感受到的动画,一方面可以是由运动的摄像机拍摄静止物体而产生的;另一方面,也可以是由静止的摄像机拍摄运动物体而产生的。此外,物体的运动不局限于空间位置的变化,也可以反映物体外观随时间的变化,例如颜色、纹理、形状等。

人之所以能够感受到连续的动画效果,主要原因在于人眼的视觉暂留机理。该机理是指视网膜形成一幅图像后,大约在 0.05~0.1s 的时间内不会消失。连续动画的基本单位是单幅静态画面。通常情况下,单独的一幅画面称为一帧,而每秒钟的帧数则用来表示动画播放的速度,称为帧率。在传统动画制作中,"一拍三"是动画的极限形式,也就是同样内容的帧画面至多重复 3 次进行播放,从而能够满足视觉暂留机理的基本要求。虽然这种方式能够很容易地实现动画效果,但是画面观看的整体体验较差。事实上,在从传统动画到计算机动画的演变过程中,涌现出了很多新的动画制作技术以用于提高动画制作水平,使其更好地满足视觉体验。

9.1.1 传统动画

如图 9.1 所示,传统的动画制作最早是采用手工绘制的方式,一帧一帧地去绘制每一帧画面。这种方式可以充分发挥动画师的创作灵感,使其能按照主观意图进行制作,因此具有很大的灵活性。但是,整个创作过程费时费力。1914 年,出现了采用分层的动画技术,首先制作关键帧,然后再生成中间过渡的帧。这个过程一般是由专业动画师用画笔在

专业的、透明度高的纸上进行绘制。随后，将多张图纸拍成胶片放入电影机播放出连续的画面。这种分层和关键帧制作的方式，能够一定程度上提高动画素材的利用率，方便动画内容的再创作。

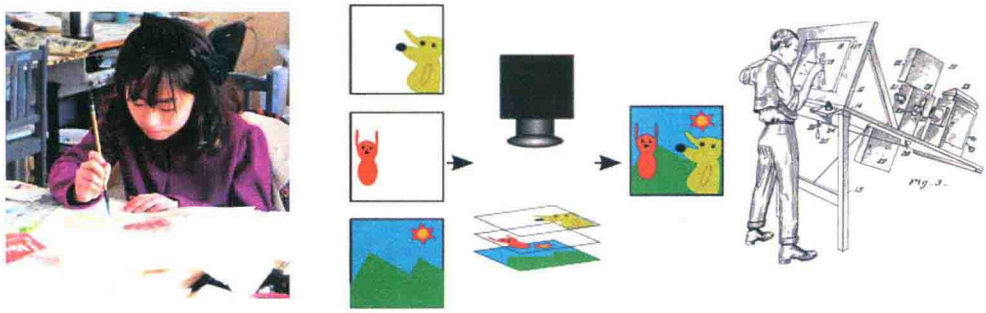


图 9.1 传统动画制作的三种方式：手工绘制、分层绘制、影像临摹

为了进一步提高动画制作效率，影像描绘的方式被引入至动画制作过程中。这种方式又称为动态遮罩或影像描摹技术，通常由动画师将真实拍摄影片中的运动逐帧地跟踪描绘出来。典型的方法是将录制好的影片的图像，投影在一个表面比较粗糙的玻璃面板上，然后由动画师按照投影出的画面进行描绘。这种用来投影生成动画的技术称为影像描摹(Rotoscope)。由于使用手绘影片的投影做参考，可以大大节省动画制作时描绘每一帧画面的时间。

9.1.2 计算机动画概述

随着计算机的出现，计算机动画开始发展起来。计算机动画是利用图形与图像处理技术，借助编程或动画制作软件生成一系列连续的帧画面。然后，通过连续播放这些单张图像的方式，产生场景和物体运动的效果。在计算机动画中，动画的表现形式可以有物体位置、方向、大小、形状、表面纹理、颜色等的变化，以及虚拟摄像机的运动。这些变化就产生了当前帧相对于前一帧部分内容的不同，从而引起视觉上的连续变化。因此，计算机生成的动画可以展现更加丰富的动画效果。

在计算机图形学兴起之前，已经存在一些利用电子计算机生成运动画面的工作。20世纪 50 年代，美国人 John Whitney 采用模拟信号的电子计算机进行光线和物体的控制，在显示屏上生成了运动的画面。因此，他也被认为是计算机动画的先驱者之一。但是，这种动画的内容仅局限于特定物理规律下的粒子运动，没有对任何现实场景画面的展现。1957 年，美国国家标准技术局的 Russell Krisch 等人在数字计算机 SEAC 上将一张拍摄的照片扫描成在屏幕上显示的画面。这也是第一张数字图像。这张图像记录了他三个月大的儿子，虽然图像的尺寸只有 176×176 像素，但也预示了使用计算机生成动画图像时代的来临。

进入 20 世纪 60 年代后，随着计算机图形学技术的兴起，计算机动画迎来了新的发展时期。图 9.2 展示了若干经典影视动画的剧照。1972 年，好莱坞影片 *Future world* 中使用了三维线框模型制作画面内容，成为最早的三维计算机动画电影(如图 9.2 所示)。

1973年，美国米高梅公司发行了第一部采用计算机动画处理的电影 *West world*，用于其中部分画面内容的制作。1975年，第二部使用三维线框的动画短片 *Great* 获奥斯卡奖。进入20世纪90年代以后，三维计算机动画迎来了辉煌时期。1995年，Pixar公司 *Toy* 是世界上第一部完全使用计算机动画技术制作的长篇动画电影，获得了奥斯卡特别成就奖、最佳原创剧本提名、最佳原创音乐提名，以及金球奖音乐喜剧类最佳影片提名、最佳原创歌曲提名等奖项。这部动画电影拿下了当年美国本土票房的冠军，也为电影制作开辟了一条新的道路。1998年，Pixar公司动画短片电影 *Geri's Game* 也获得了奥斯卡最佳短片奖。值得一提的是，该影片中动画主角的人体三维模型，完全是采用第三章中所介绍的Catmull-Clark细分曲面进行制作，能够生动地表现出人物角色的各种姿态和表情。

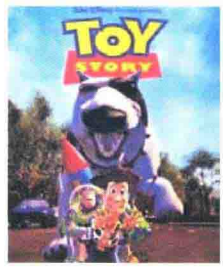



图 9.2 电影 *Future world*、*Toy* 和 *Geri's Game* 剧照

9.1.3 动画制作流程

不论是传统动画，还是计算机动画，在制作时大都遵循相似的流程。通常一个完整的动画作品可以看作由帧、镜头、序列和最终作品形成的四级层次结构组成。在动画制作流程的不同阶段，完成相应层级的内容制作，最后组装起来形成完整的动画。

首先，通过故事展板的形式将动画情节进行编纂，构建整个动画内容的情节发展主线。这就需要提出创意，提供符合主线的内容概要，并形成动画剧本。然后，对于动画中的角色进行建模，例如采用二维图像或者三维模型等形式，表现角色的各种姿态和表情。接着，一般由艺术家或者经验丰富的动画师创作关键帧的画面，使其能够合理地呈现动画情节。在此基础上，再创作关键帧之间的过渡帧，形成连续的动画帧画面。随后，将情节关联的一系列连续帧组成镜头。最后，对角色进行上色、声音添加等加工，完成整个动画作品。

在计算机动画出现以后，动画制作过程的变化主要体现在关键帧之间的过渡帧，逐渐由计算机程序生成取代手工制作。关键帧是指动画序列中展现对象特定时刻形态的关键画面，例如动作的起始和终止画面。关键帧的概念最早起源于早期迪士尼卡通画的制作。此外，镜头测试、上色等环节，也可以通过计算机局部变形或者图形绘制技术生成。这些都大大提高了动画制作的效率，同时降低了制作成本。

在动画制作流程中，如何由关键帧生成中间过渡帧，是计算机动画制作时非常重要的环节。常见的有关键帧插值、过程建模和物理模拟等三种技术。另一种生成动画的方式是对角色模型进行直接控制，例如骨架驱动和运动捕捉，从而得到各种不同的姿态。不同

的技术各有优劣，接下来做具体介绍。

9.2 关键帧插值

如图 9.3 所示，动画可以看作是在关键帧的基础上，对中间时刻的形态进行插值来生成渐变的连续画面。在早期的动画制作室里，往往是由高级动画师设计卡通片中的关键帧，然后再交由助理动画师设计中间帧。起初的关键帧技术仅仅用来插值帧与帧之间卡通画角色的外观，不久后便发展成为可以用来插值影响运动的任何参数。这样能够更好地控制中间帧的变化。关键帧技术可以使动画师只需设计起始和终止的两帧画面，而中间帧则完全由计算机自动生成，从而大大简化动画制作的过程。

图 9.3 关键帧插值技术

在关键帧插值时主要考虑两方面的因素：一是在什么定义域上对物体形态进行插值，比如颜色的插值、形状的插值等；二是要考虑插值后中间状态变化的连续性，也就是插值过程的连续性，例如线性插值、样条插值等的连续性。

交融技术(Cross dissolve)，或称为淡入淡出技术，是对颜色进行线性插值的常用方法。这种方法是将一幅画面的颜色进行淡出处理的时候，逐渐淡入另一幅画面的颜色，从而获得两幅画面之间颜色的渐变。但是，交融技术没有考虑形状的对齐，往往导致渐变时的视觉效果不佳。相比而言，形状插值及其连续性是在关键帧插值时更为困难的问题。

在关键帧的形状插值时，线性插值和样条插值是两种最基本的关键帧插值方式。这两种方式的基本思想都是通过对位置决定的运动轨迹进行间隔采样，获得其在每一帧上的位置。这些通过计算所得到的位置就决定了中间形状。假设物体表示为顶点及其连线组成的网格，那么两个关键帧上相对应的顶点 $\boldsymbol{v}_i^s$ 和 $\boldsymbol{v}_i^e$，在 t 时刻的位置可以通过对这两个顶点做线性插值进行计算，记为：

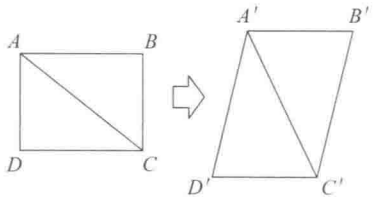
$$\boldsymbol{v}_i^t = \frac{t-s}{e-s}\boldsymbol{v}_i^s + \frac{e-t}{e-s}\boldsymbol{v}_i^e \tag{9.1}$$

线性插值虽然简单，但也存在很多问题，例如真实度不高，这是由于大部分物体不是以直线形式运动的，而且大部分运动随时间的变化也不是线性的。此外，对于位置连续的运动，速度也可能会产生不连续的变化。样条插值是用户先设计好物体运动轨迹的关键位置点，然后利用样条曲线进行拟合。常用的样条曲线包括 Hermite 曲线、Bézier 曲线、B 样条曲线等。相比于线性插值，样条插值的结果具有更高的连续性。但是，由于插值的形状仅表示为顶点位置，插值出来的中间形状仍不自然。因此，在插值时需要进一步考虑其

他形状因素。接下来主要介绍二维图像的形变插值和三维图形的形变插值，部分内容也可参考第4章中的网格形变。

例题 9-1 样条插值关键帧。

问题：右图所示的初始和结束形状，分别用四个顶点记为(A,B,C,D)和(A',B',C',D')。写出采用线性插值和B样条曲线插值顶点位置来生成中间形状的伪代码。



解答：假设线性插值和B样条曲线插值时按照匀速的参数变化生成中间形状的顶点位置，那么伪代码如下。

```
function keyInterpolation (A, B, C, D, A', B', C', D')
    start[4] ← {A, B, C, D}                    /* 数组记录起始形状位置 */
    end[4] ← {A', B', C', D'}                  /* 数组记录终止形状位置 */
    for i ← 0 to 3 do
        L[i] ← line (start[i], end[i])         /* 计算连接对应顶点的线段 */
        B[i] ← spline (start[i], end[i])       /* 计算连接对应顶点的 B 样条曲线 */
    end for
    for i ← 0 to 3 do
        for t ← 0 to 1 do
            SL[i](t) ← L[i](t)                 /* 沿着线段插值的中间形状 */
            SB[i](t) ← B[i](t)                 /* 沿着 B 样条曲线插值的中间形状 */
        end for
    end for
end function
```

□

9.2.1 形状保持的图像形变插值

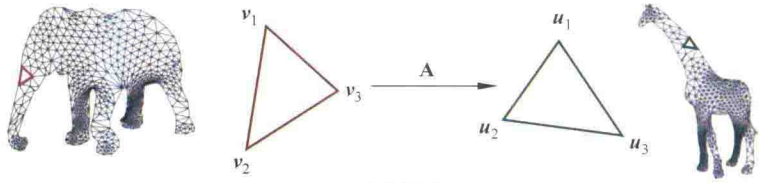
在关键帧插值时，通常需要在形变过程中尽可能地保持形状的局部几何特征。对于平面形状而言，平移变换、刚性变换和相似变换是能够保持形状局部特征的基本几何变换。因此，可以利用这些变换进行物体形变，从而获得中间帧上的物体形状。

这里采用平面三角网格表示每一帧图像中的物体形状，也就是对物体的轮廓进行三角剖分(如图9.4(a)所示)。那么，根据网格顶点的对应关系，可以计算每一个三角形从起始帧到终止帧唯一的仿射变换，记为：

$$\boldsymbol{u}_i = \boldsymbol{A} \cdot \boldsymbol{v}_i + \boldsymbol{b} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \boldsymbol{v}_i + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{9.2}$$

其中，$\boldsymbol{v}_{i=1,2,3}$和$\boldsymbol{u}_{i=1,2,3}$分别是起始帧和终止帧上对应三角形的三个顶点(如图9.4(a)所示)。这样，在起始帧上用三角形表示的局部形状特征，就可以通过仿射变换映射到终止帧上对应的三角形。进一步，利用上述变换生成中间帧上的中间形状。这是需要对其包含的刚体变换和相似变换分别做插值计算。一种方式是直接对矩阵$\boldsymbol{A}$的元素进行插值，也就是按照对应元素位置上的数值进行插值。但是，这样很容易造成中间状态的形状萎

缩。另一种方式是采用矩阵运算方式来实现矩阵插值。这需要对仿射变换矩阵进行极分解(Polar decomposition)操作,将仿射矩阵表示为刚性变换矩阵和相似变换矩阵的乘积形式,然后分别对这两个矩阵进行插值。



(a) 三角形变

(b) 形状插值结果

图 9.4 形状保持的图像形变插值(图片来自[113])

数学上,可逆矩阵经过极分解可以表示为正交矩阵和对称正定矩阵的乘积。具体来讲,通过极分解可以将仿射变换矩阵 $\boldsymbol{A}$ 表示为 $\boldsymbol{R}^{\theta} \cdot \boldsymbol{S}$。其中,正定矩阵 $\boldsymbol{R}^{\theta}$ 对应于旋转角度为 θ 的刚性变换矩阵,记为

$$\boldsymbol{R}^{\theta} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{9.3}$$

对称正定矩阵对应于相似变换矩阵,记为

$$\boldsymbol{S} = \begin{bmatrix} s_x & s_h \\ s_h & s_y \end{bmatrix} \tag{9.4}$$

其中,s_x 和 s_y 表示了沿两个方向的伸缩量。那么,在中间时刻 t 的形变矩阵就可以通过如下矩阵插值公式进行计算:

$$\boldsymbol{A}_t = \boldsymbol{R}^{t\theta} \cdot ((1-t)\boldsymbol{I} + t\boldsymbol{S}) \tag{9.5}$$

这里 $\boldsymbol{I}$ 是单位矩阵。公式(9.5)实际上是对刚性变换的旋转角度和相似变换的伸缩量进行线性插值,从而生成中间帧上对应的变换矩阵 $\boldsymbol{A}_t$。其中,对应于中间帧上的旋转变换 $\boldsymbol{R}^{t\theta}$ 表示为

$$\boldsymbol{R}^{t\theta} = \begin{pmatrix} \cos t\theta & -\sin t\theta \\ \sin t\theta & \cos t\theta \end{pmatrix} \tag{9.6}$$

对应于中间帧上的相似变换 $t\boldsymbol{S}$ 表示为

$$t\boldsymbol{S} = \begin{bmatrix} ts_x & ts_h \\ ts_h & ts_y \end{bmatrix} \tag{9.7}$$

然后将两者相结合,就得到矩阵运算插值的结果。

上述矩阵运算的插值过程,反映了在起始和终止时刻之间的中间帧上形状所对应的变换。通过将这种插值方式得到的变换施加到起始形状,就可以得到对应的中间形状,并

且能够更好地保持物体形状的局部几何特征，也就是每个三角形的形状（如图 9.4(b)所示）。这里需要注意的是，由于中间帧上的变换存在相似变换的分量，可能导致插值出的中间形状存在一定程度的各向等比例缩放。这在进行较大尺度的形变时，会影响插值所得到的中间形状的合理性。

例题 9-2 形状保持的形变插值。

问题：假设例题 9-1 所示的起始和终止帧的两个形状，写出通过形状保持的形变插值来生成中间帧形状的伪代码。

解答：假设两个形状对应的三角形表示分别记为$\{\triangle ABC, \triangle ACD\}$和$\{\triangle A'B'C', \triangle A'C'D'\}$，那么形状保持的形变插值的伪代码如下

```
function shapeInterpolation (A, B, C, D, A', B', C', D')
    start[4] ← {A, B, C, D}                          /*数组记录起始形状位置*/
    A1 ← getAffine (△ABC, △A'B'C')                   /*计算仿射矩阵*/
    (R1, S1) ← polarDecomp (A1)                      /*极分解得到旋转和缩放变换*/
    A2 ← getAffine (△ACD, △A'C'D')
    (R2, S2) ← polarDecomp (A2)
    for t ← 0 to 1 do
        A1 (t) ← R1 (t)S1 (t)                        /*分别插值旋转变换和缩放变换*/
        A2 (t) ← R2 (t)S2 (t)
        for i ← 0 to 3 do
            if start[i] in △ABC then
                start[i](t) ← A1 (t) · start[i]      /*形变插值变换顶点位置*/
            else
                start[i](t) ← A2(t) · start[i]       /*形变插值变换顶点位置*/
            end if
        end for
    end for
end function
```

□

9.2.2 形状保持的三维网格形变插值

在三维动画过程中，往往在对三维网格表示的形状进行形变时，控制其局部形状扭曲尽可能的小。同时，中间帧上对应的插值的中间形状要有较高的连续性，从而获得连续的动画序列。借鉴形状保持的图像形变插值方法，可以对三维网格发生形变时的每个三角形，添加局部刚性变换约束，从而获得能够连续变化的旋转变换。需要注意的是，这里的变换是对应于三角形在三维空间的旋转变换。

假设生成中间形状的形变函数记为 F，那么对于三角网格上所有三角形的顶点$\{\boldsymbol{v}_i\}$，相应的旋转变换 $\boldsymbol{R}_i$ 应该满足如下公式：

$$\underset{\boldsymbol{R}_i \in SO(3)}{\operatorname{argmin}} \sum_{\{\boldsymbol{v}_i\}} \left(\| \mathrm{d}F - \boldsymbol{R}_i \|_F^2 + \alpha \| \mathrm{d}\boldsymbol{R}_i \|_F^2 \right) \tag{9.8}$$

这里 $SO(3)$是所有三维旋转变换构成的几何变换群，而 α 是和局部形状有关的权因

子。该权因子用于控制局部形状的保持程度和相邻局部的连续性。因此，公式(9.8)中的第一项包含了局部刚性变换的约束，而第二项则包含了邻域变换的连续性约束。

对应于三角网格，通过对起始帧和终止帧上对应顶点 1-邻域变换的极分解，可以将公式(9.8)进一步转化为如下表达式：

$$\min_{\boldsymbol{R}_k \in SO(3)} \sum_{k=1}^{m} \sum_{\boldsymbol{v}_j \in N(\boldsymbol{v}_k)} \| \boldsymbol{u}_{ij} - \boldsymbol{R}_k \cdot \boldsymbol{Y}_k \cdot \boldsymbol{v}_{ij} \|^2 + \hat{a}_k \sum_{\boldsymbol{v}_j \in N(\boldsymbol{v}_k)} w_{kl} \| \boldsymbol{R}_k - \boldsymbol{R}_l \|^2 \tag{9.9}$$

其中，$\boldsymbol{v}_{ij}$ 和 $\boldsymbol{u}_{ij}$ 是分别以顶点 $\boldsymbol{v}_i$ 和 $\boldsymbol{u}_i$ 为原点的 1-邻域局部坐标系下的位置，$\hat{a}_i$ 是顶点 $\boldsymbol{v}_i$ 的 1-邻域面积(如图 9.5(a)所示)。公式(9.9)中的 $\boldsymbol{Y}_k$ 对应于极分解后的尺度缩放变换。这里的极分解与公式(9.3)和公式(9.4)类似，只是需要对三维空间的变换矩阵进行相应的分解操作。

通过优化上述公式，便可以得到生成中间形状所对应的旋转变换。进一步，再结合极分解得到的相似变换来计算插值变换，最终便可以获得中间帧上所对应的中间形状。图 9.5(b)展示了根据首尾两个形状的三角网格计算得到的中间形状插值结果。

(a) 局部 1-邻域　　(b) 形状插值结果

图 9.5　形状保持的三维网格形变插值(图片来自[114])

9.3　物 理 模 拟

如图 9.6 所示，基于物理模拟的动画制作，是通过对动画产生的物理过程进行模拟，从而生成连续变化的帧序列，并使其具有客观世界中物体运动所满足的物理规律。简而言之，就是要从物理学角度研究计算机动画中物体的运动规律。粒子系统是一种典型的物理模拟生成动画的方法，是影视特技中生成特殊视觉效果的一种主要方法，也是最实用的动画技术之一。

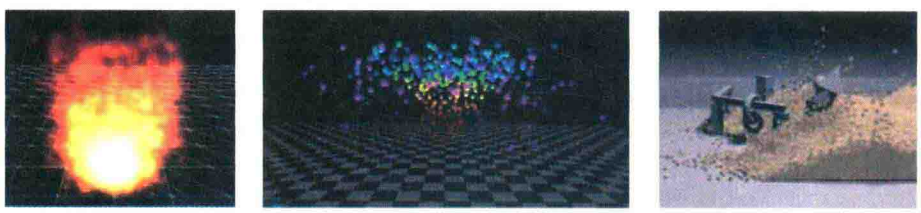


图 9.6　粒子系统的动画效果

粒子系统在表示物体形状方面更加灵活，可以用于解决自由曲面或网格无法有效表示不规则或随机变化形体的问题，例如云、雾、水、火、烟等。1983 年，Lucas 影业公司的

William Reeves 在他的论文中首次提出了一种模拟不规则自然景物造型和动画的系统，也就是所谓的粒子系统。该系统采用了一套完全不同于以往几何建模和绘制的方法来重新构造和绘制场景，能够同时实现多种动画效果。

粒子系统采用粒子作为基本单位，将场景定义为由成千上万个、形状不规则、位置随机分布的粒子所组成，而每个粒子均具有一定的生命周期。粒子在生命周期内不断地改变形状、不断地在空间做各种运动。常用的粒子表示有点、球、椭球、立方体、圆等基本几何形状。

采用粒子系统生成动画的基本原理，是通过在生命周期内控制粒子随时间的状态变化来达到相应的动画效果。粒子在生命周期内主要有三种状态的变化：新生态、成长态和死亡态(如图 9.7 所示)。新生态，是刚创建的粒子所处的初始状态，实际上包含了粒子的创建和初始化两个过程，同时也定义了粒子的基本属性；成长态，是指在生命周期内粒子的各种属性随时间的变化状态，也就是允许粒子属性本身的各种改变，以引起粒子形态在生命周期内的变化，由此形成一系列的动画画面；死亡态，是指存在时间超过生命值后的状态，表示粒子在生命周期后的消亡，最终退出动画场景。

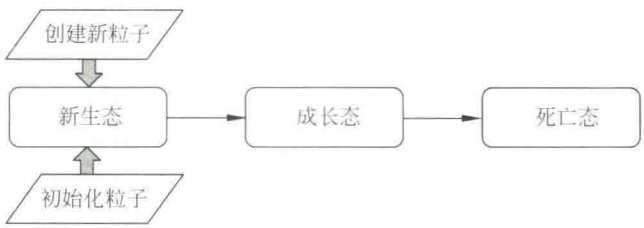


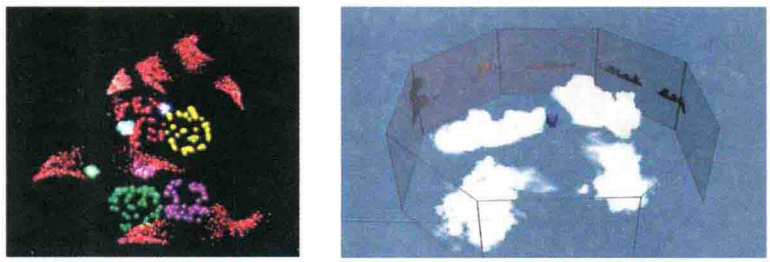
图 9.7 粒子在生命周期内的变化

此外，为了便于动画过程中对粒子的控制以及提高动画生成的效率，通常将粒子系统建立在一些约定的假设之上，以此简化计算。例如，每个粒子一般不会与其他粒子发生碰撞，由此避免了大量的求交运算；粒子本身不反射光，在非聚集状态下，不会向其他粒子投射阴影，而只会向场景的其他环境投射阴影，由此避免了大量的阴影计算。

粒子系统中的每个粒子在状态转变的过程中，均经历了出生、成长和死亡的阶段，与粒子有关的每一个参数均受到指定的随机过程的控制，以使得每个粒子能够表示不同的运动。在出生阶段，对于某一瞬间的画面，新的粒子会根据一个控制的随机过程加入系统中，而用户可以通过画面中的平均粒子数及其分布建立系统的初始状态。例如，画面中出现的粒子总数可以设为 $a+\sigma\times \mathrm{rand}(t)$，其中 a 是平均粒子数，σ 是方差，而函数 rand(·) 是关于时间 t 的随机函数。此外，这个阶段也会赋予每一个粒子相应的属性，包括位置、速度、大小、质量、颜色、透明度等。

在成长阶段，每个粒子根据其初始状态和随机过程，通过属性的变化产生新的状态，以此得到一帧新的画面。例如，对于位置和速度，可以按照物理上的牛顿定律，根据粒子受力情况，计算加速度，进而更新速度和位置；颜色和透明度，则可以设置为时间的函数，按照一定的绘制方式获得粒子的状态。事实上，粒子系统的绘制方式多样且灵活，可以把每个粒子作为一个光源，然后设置那些可以照亮的像素的颜色(如图 9.8(a)所示)。此

外，粒子的绘制方式还有纹理广告牌(Textured billboard)。它是将粒子的聚集效果作为纹理贴图进行绘制(如图 9.8(b)所示)。这里需要注意，虽然粒子之间的碰撞可以被忽略，但在成长阶段，粒子和周围环境还是要发生碰撞，这就不可避免地影响最终的绘制效果。因此，如何有效地进行碰撞检测和求交计算，也是采用粒子系统生成动画的重要问题。



(a) 粒子光源　　(b) 纹理广告牌

图 9.8　粒子系统的绘制方式(图片来自[115]和[116])

在死亡阶段，系统会删除那些已经超出其生命周期的粒子。这里超出生命周期主要有三种情况：生命周期结束、淡出和移走。由于每个粒子在出生时都被赋予了一个生命周期，每次生成动画的一帧画面时，粒子的生命值就会减 1。那么，当生命值变为 0 时，说明该粒子应当被删除。当粒子的颜色或透明度小于给定的阈值时，说明该粒子不可见，也需要被删除。此外，当粒子的运动超出了画面范围时，也是不可见的，需要被删除。

为了在上述的三个阶段中有效表示粒子及其状态变化，需要设计合理的数据结构。粒子系统中的粒子往往具有很大的随机性，通常采用链表的数据结构表示粒子。由于链表是一种物理存储单元上非连续、非顺序的存储结构，在进行数据的插入、访问等操作时能够达到低于线性复杂度的效率，因此非常方便管理每个粒子出生、成长和死亡的过程。

9.4　运动捕捉

运动捕捉(Motion capture，Mocap)是通过软/硬件方式记录、分析并处理人或其他物体的动作，以此驱动计算机中各种动画角色做出相应动作的技术，也被称为动态捕捉、运动跟踪等，是计算机动画制作中非常重要的技术。一般而言，运动捕捉涉及尺寸测量、空间定位等方面的技术，其主要目的是要测量、跟踪、记录物体在三维空间中的运动轨迹，从而获得可以被计算机处理的三维空间数据，进一步在计算机中用于动画制作的过程。

运动捕捉起源于传统动画生成技术中的影像描摹，最早是在 1915 年由美籍波兰人费舍尔(Fleischer)发明。目前的运动捕捉系统主要分为主动式和被动式两类，各自使用不同的传感器、信号捕捉设备、数据传输设备、数据处理设备实现人体动作的获取。主动式是系统本身发射信号用于探测肢体做出的各种动作，需要依附在肢体上的标记进行记录，例如图 9.9 所示的采用电子机械式、光纤式、闪光标记式等手段。被动式的系统本身不发射信号，无须做任何标记，例如仅通过视频拍摄进行动作捕捉。这两类方式在动作捕捉时

的效果各有优劣，在实际中都有着广泛的应用。

图 9.9 主动式运动捕捉的三种方式：电子机械式、光纤式、闪光标记式

9.4.1 主动式运动捕捉

1. 电子机械式

电子机械式是通过机械装置将传感器附着在肢体上的主要关节点处，形成连接在一起的测量结构。这种结构往往是由多个关节和刚性连杆组成的，同时在可转动的关节处装有角度测量装置，可以测得关节转动的速度、角度等的变化情况。这种方式的优点是能够实时捕捉动作，不存在肢体部位之间的遮挡问题，而且可以捕捉较大范围的动作，但使用起来不够方便。这主要是由于机械结构往往对使用者的动作阻碍和限制很大，导致使用者很难自由地运动，使得能够准确捕捉的动作类型有限。此外，这种方式在运动捕捉时采样率较低，精度也较低，大多用于静态造型捕捉和关键帧的确定。

但是，电子机械式是在电影制作中是最早使用的运动捕捉系统。典型的例子是好莱坞电影《侏罗纪公园》的拍摄，其中各种恐龙行走、奔跑的动作，就是通过捕捉人的相应的运动来获取的。然后，计算机生成的恐龙模型再按照这些捕捉到的动作生成连续的动画序列。

2. 光纤式

光纤式主要应用于数据手套，是在数据手套中沿手指等运动关节布置光纤传感器。当手指弯曲时，驱使光纤弯曲，从而造成透射光衰减。那么，基于衰减的光的强度，就可以准确测量手指关节旋转角度。光纤式的优点是没有遮挡，能够进行实时捕捉，而且可捕捉小范围的弯曲等运动。但缺点是需要根据个体对手套尺寸进行调整，以满足不同人的使用。而且，光纤式捕捉精度较低，只适用于手部动作的获取。

3. 闪光标记式

闪光标记式是通过在较暗光照下拍摄附着在肢体上的闪光点跟踪运动状态。这里，通常在运动物体的一些关键部位（例如人体的各个关节处），使用一组按顺序接通的 LED 灯作为标记点，以便在预先设定的时间内只有其中的少数一些标记点可见。然后，通过多个摄像机从不同角度的拍摄，就可以确切知道相应标记的二维平面位置。最后，这些位置数据实时传输至工作站，利用三角测量原理就能精确计算标记点的三维空间坐标，再从运动学原理解算出三维空间的六自由度运动。闪光标记式的优点是捕捉速度快、捕捉精度

高。但缺点是只能在室内有限光亮环境下使用，而且容易发生肢体遮挡，捕捉动作时的精度受相机位置影响比较大。

9.4.2 被动式运动捕捉

如图 9.10 所示，被动式主要依赖从摄像机拍摄的图像/视频直接重建三维动作序列，也称为光学图像式动作捕捉。这类方式通常需要人工标注的数据作为训练集，在捕捉动作时通过检测、分类、识别等手段对不同类型的动作进行跟踪，以此形成连续的动作序列。光学图像式的优点是捕捉速度快、设备简单。但缺点是受光照等影响明显，而且也存在肢体遮挡、捕捉动作的精度受限等问题。

(a) 人体姿态动画　　(b) 人脸表情动画

图 9.10 光学图像式运动捕捉

在具体使用时，可以通过三种方式进行动作捕捉。第一种方式是采用普通彩色摄像机拍摄二维图像序列，检测和提取主要关节点的二维坐标，再根据多视角几何计算三维坐标。第二种方式是结合红外相机获取热光源辐射，以此提高动作捕捉时人体检测的准确性。第三种方式则是利用 Kinect 等深度相机，直接捕捉三维动作。

9.4.3 运动重定向

运动捕捉的数据提供了人体在实际运动中的运动个性和细节，尤其是关节、肌肉等的运动。这些可以很容易地被映射到虚拟角色，驱动这些角色做出人体所对应的动作，从而通过计算机生成逼真的动画。这个过程称为运动重定向(Motion retargeting)。如何有效地进行运动重定向，是通过运动捕捉进行动画制作的重要步骤。

由于运动捕捉时人体和角色在外形等方面的不匹配，对于捕捉的人体关节的运动数据，即人体姿态动画(如图 9.10(a)所示)，需要进行后期的信号处理和编辑操作，以满足重定向虚拟角色的特点。运动数据的信号处理是考虑如何通过信号频率来描述各种运动。在对运动数据做信号处理时，往往将运动看成多维信号。运动的编辑则是对运动信号进行改变，使其满足用户指定的一些约束。这样在运动重定向时，首先通过低通滤波来去除信号中的噪声。然后，利用不同频率信号分量分别刻画运动类型(低频部分)和运动特性(高频部分)。最后，通过对角色模型进行编辑操作，例如 4.6 节介绍的网格编辑方法，使其满足运动捕捉数据所呈现的运动类型和运动特性。此外，运动捕捉时往往是每次采集小的运动片段，然后再连接起来得到完整的动作序列。这就需要将不同片段的动作

按照时间串联，形成连续的运动。为此，采用线性或非线性的方式，在时间重叠部分将前后两个运动数据进行混合，以获得自然的运动过渡。

除了人体姿态动画外，人脸动画是更为重要的一个问题。这主要是由于脸部的表情可以有效地传达人物的情绪，从而更好地服务于动画内容的制作（如图 9.10(b)所示）。其中，基于光学图像式的人脸动画生成方式包括实时人脸表情跟踪、直接人脸动画捕捉等类型。

实时人脸表情跟踪大多基于二维视觉控制三维模型的思路，从拍摄的视频里面提取动画参数，然后用于三维人脸模型，进而生成三维的人脸动画序列（如图 9.11 所示）。这种思路需要一段记录人脸表情活动的单个视频帧序列和一个通过激光扫描获取的三维人脸模型作为输入，同时需要一个预处理的头部运动捕捉数据库，然后通过四个步骤生成三维人脸动画。第一步，通过跟踪视频中头部运动进行画面分析，主要是获得表情控制和头部姿态参数。第二步，结合头部运动数据库将表情控制参数进行抽取，获得视频帧画面记录的表情描述。第三步，对连续帧的表情控制参数进行滤波去噪，获得高质量的表情活动。第四步，根据表情控制参数驱动高精度三维人脸模型，使其产生符合视频画面内容的表情动画。这种方式可以采用低分辨率的视频驱动高精度的三维人脸模型，从而获得高质量的人脸动画。

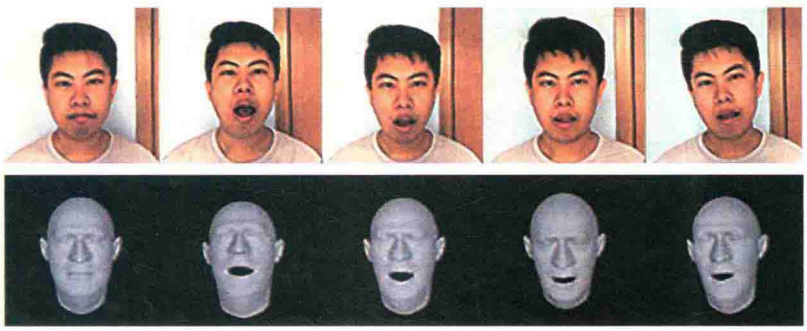


图 9.11 基于单个视频实时人脸表情跟踪的动画（图片来自[117]）

直接人脸动画捕捉则是采用高精度三维扫描和多相机阵列直接获取人脸的几何、颜色、光照等信息，进而通过动态纹理贴图在三维人脸模型上重建人脸动画序列。这里可以使用六个标定好的相机同时记录不同视点位置的脸部活动，并通过跟踪特征点来驱动三维人脸模型的几何变形。此外，通过对不同视点的画面进行视角插值，获得动态纹理贴图，映射到变形后的三维人脸模型，可以生成更加真实的表情动画。

9.5 群体动画

在生物界，诸如鸟、鱼等许多动物是以某种群体的方式运动的。在动画设计和影视特技中，不可避免地会遇到各种大规模群体动作场面的制作。如图 9.12 所示，两军对战中的数十万大军冲锋的效果，兽群、鸟群集体活动等。这种运动既有随机性，又有一定的规律性，但是难以基于单个物体动画的集合来描述。美国 Symbolics Graphics 公司的 Craig

Reynolds 系统性地提出了群体动画(又称为群组动画)技术,并成功地模拟了这种方式的运动。一般来说,群体动画主要解决的问题包括个体行为真实性和整体视觉质量。前者忽略个体的视觉效果,在精确地定义个体行为后,通常借助简单的二维表示来快速生成动画过程;后者则使用高效的真实感绘制和仿真技术来满足高质量的整体视觉感受。

图 9.12 影视制作中的群体动画案例

Craig Reynolds 认为群体的行为包含两个对立的因素,也就是既要相互靠近,又要避免碰撞。他用三条按优先级递减的原则来控制群体的行为:碰撞避免原则,也就是避免与相邻的群体成员相碰;速度匹配原则,也就是尽量匹配相邻群体成员的速度;群体合群原则,也就是群体成员尽量靠近。因此,群体动画一个很重要的任务是模拟群组的运动。群组是指在同一物理环境下拥有相同目的的一群个体,他们的行为有别于作为单独个体成员时的行为。这样在描述群体动画时,就需要对群(Crowd)、组(Group)、个体(Individual)三个层次的行为进行建模、绘制,通过这三个层次内容的有机结合生成相应的动画序列。在定义了场景中的群、组、个体三个层次的内容后,群体动画的关键问题就是如何有效模拟自然的群组运动。这既需要保持群体中个体运动的集体表现,也需要避免个体之间的碰撞。

一般而言,群组具有数据量大、计算量大的特点,而其运动兼具随机性和规律性。目前,主要有宏观和微观两种方式来控制群组运动。宏观方式将所有个体始终作为一个整体进行建模,采用流体或气体动力学描述群体密度和运动速度的改变。这样,群体的运动效果就是通过求解偏微分方程来获得。微观方式则是需要对个体行为进行建模,不仅考虑个体本身的运动属性,还要计算个体之间的相互作用,以此来准确地模拟群体动画。其中,行人的群体运动又是影视动画、计算机游戏中最为常见的内容,典型的场面有城市街道的人群、战场上的士兵、大型演出的观众等。这类群体运动通常采用向量场的连续插值来得到整体运动,再通过随机个性化的运动参数调节,以使得每个行人又表现出不同的运动姿态。

总体而言,群体动画的目的是为了生成更加壮观、自然的场景,从而有效地提升大场面的视觉震撼效果。这不仅可以减少动画师的工作量,而且能大大节约成本。典型的应用有《指环王》系列、《狮子王》《纳尼亚传奇》《星球大战前传》《木乃伊》等影视作品。接下来介绍两种代表性的群体动作模型:Flock-and-Boid 模型以及社会力模型。Flock-and-Boid 模型是针对动物群体行为的模型,而社会力模型则是专门面向人群的动画制作模型。

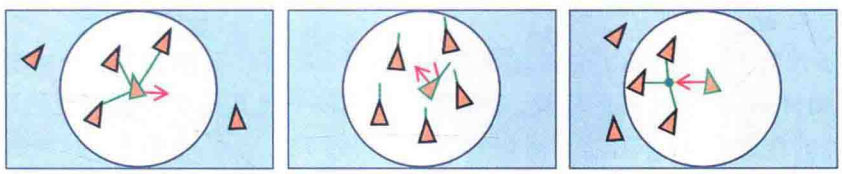
9.5.1 Flock-and-Boid 模型

事实上，包含大量粒子运动的粒子系统，是可以被用来描述群体动画中的群组运动。然而，由于粒子的基本形状过于简单，导致其只能模拟简单的群体行为。一种直接的改进方式是把粒子系统变得更加复杂。为此，Craig Reynolds 提出了 Flock-and-Boid 模型，通过对形状、行为等更准确的建模，实现符合群体行为的动画效果。

Flock 是具有整体对齐、非碰撞、聚集运动的一组物体，而 Boid 则是指模拟类似于鸟、鱼等的物体。Flock 可以看作不同 Boid 的个体行为之间相互作用后的整体结果。这样，就可以在模拟个体简单行为的基础上，通过个体之间的互动来呈现群体行为。从这个意义上讲，每个 Boid 就对应于单个粒子，而 Flock 则是整个粒子系统。下面以鸟群为例，分别介绍 Flock-and-Boid 模型中对 Boid 和 Flock 的建模方式。

每一个 Boid 个体应该具有鸟的大致形状，而不是简单的点、球、立方体等粒子的基本形状。此外，个体在飞行时沿着一条路径进行动态、增量和刚性运动，同时允许其形状在飞行坐标系中任意改变形状，例如挥动翅膀、摇摆头颈等。这里的路径是三维空间中的一条曲线，而飞行坐标系是以每个 Boid 中心为原点的局部坐标系，这样也方便对个体的物理仿真。例如在飞行过程中，个体应当遵循动量守恒、速度上下限、最大加速度、以及重力作用下的身体倾斜等约束，以使得个体行为的描述更为准确。这些都可以作为粒子系统中的粒子属性进行设置，而 Boid 和粒子最大的区别则是在于要考虑不同个体之间的作用。

具体来讲，鸟群中的每一只鸟在飞行时都要允许与它的同伴进行协调，体现在鸟群中互相靠近的趋势，同时又不能发生个体碰撞。这是通过对 Boid 个体引入局部感知来实现。如图 9.13 所示，这种局部感知主要包含个体对飞行位置和速度两方面的控制：位置上既要合群又要避免碰撞；速度上要尽量匹配邻近的其他个体。这也就是群体行为应该满足的三项基本原则。



(a) 避免碰撞　　(b) 速度匹配　　(c) 群体合群

图 9.13　Boid 个体的局部感知

根据上述个体的运动行为，所有 Boid 的集合就构成了 Flock，并形成整体的群体行为模型。这个过程也称为合群，遵循以下基本的合群原则：Boid 局部感知的群体中心是相邻个体子集的中心；越靠近群体边缘的 Boid 个体，受局部感知的影响越大；群体允许分裂和合并。而对于 Flock 整体的行为，通常是设置整体的飞行目标、方向、路径变化等方式进行控制。在此基础上，Boid 也会做出相应的变化，以满足 Flock 的群体行为。图 9.14 展示了采用 Flock-and-Boid 模型实现鸟群动画的一帧画面。

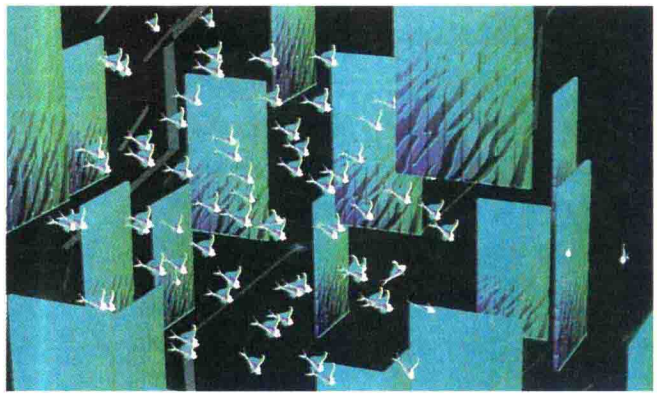


图 9.14 基于 Flock-and-Boid 的鸟群动画(图片来自[119])

9.5.2 社会力模型

与动物群体的行为不同,人群聚集时所产生的群体行为除了受到环境中的物理约束外,还会受到社会心理学影响,尤其是在发生恐慌事件时。当人处于恐慌状态时,个体的行为动作会变得迟缓和不理智,产生所谓的"羊群效应"。事实上,获取真实的群体恐慌行为数据是非常困难的。这就需要通过群体动画来模拟人的群体行为。2000 年,匈牙利和德国的科学家合作在《自然》杂志上发表了一篇关于人群行为模拟的文章,提出了一种"社会力模型"。该模型能够对人群聚集环境下的群体行为进行有效建模,并通过动画展现恐慌状态下的群体行为。

简单地讲,社会力模型是基于牛顿力学来描述受社会心理学影响的个体之间的相互作用。社会力作用下的个体行为受主观心理、其他个体和环境三方面因素的影响,都可以等效为力在个体上的作用,其力学模型可以表达为如下形式:

$$m_i \frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = F_i + G_i + H_i = m_i \frac{v_i^0(t)\boldsymbol{e}_i^0(t) - \boldsymbol{v}_i(t)}{\tau_i} + \sum_{j \neq i} f_{ij} + \sum_w f_{iw} \tag{9.10}$$

这里 $\boldsymbol{v}_i$ 表示作为个体的人在 t 时刻运动的实际速度,v_i^0 和 $\boldsymbol{e}_i^0$ 分别表示初始速度大小和方向,τ_i 是运动时间,m_i 是个体的质量。个体在运动过程中受其他个体及环境的影响,体现在相邻个体的作用力 f_{ij} 和环境中障碍物等的作用力 f_{iw}。那么,个体在 t 时刻的实际位置 $\boldsymbol{r}_i(t)$ 则可以通过对速度的积分进行计算,也就是 $\boldsymbol{r}_i(t) = \int_0^t \boldsymbol{v}_i(t)\mathrm{d}t$。

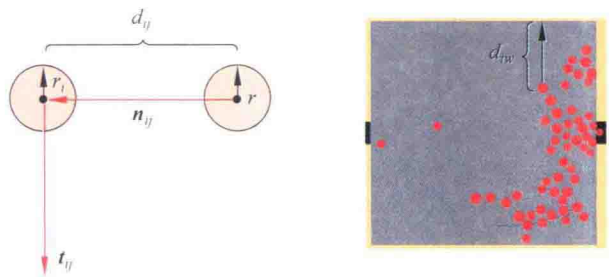
在公式(9.10)中,F_i 项反映了主观心理对于个体行为的影响,体现了个体在 τ_i 时间段内以期望的速度朝着目的地运动的动机,其物理原理是牛顿力学第二定律。

G_i 项则是反映了周围其他个体的影响。如图 9.15 所示,在社会力模型中,每个行人个体都是用半径不同的圆来描述。这样,其他个体所施加的影响具有如下形式的力学公式:

$$f_{ij} = \{A_i \exp[(r_{ij} - d_{ij})/B_i] + kg(r_{ij} - d_{ij})\}\hat{\boldsymbol{n}}_{ij} + kg(r_{ij} - d_{ij})\Delta v_{ji}^t \boldsymbol{t}_{ij} \tag{9.11}$$

其中,第一项表示了个体之间的排斥力(又称为心理学力),它是跟个体之间的距离 d_{ij} 以及个体所占区域圆半径之和 $r_{ij} = r_i + r_j$ 有关(如图 9.15(a)所示),同时 $\boldsymbol{n}_{ij} = (\boldsymbol{r}_i - \boldsymbol{r}_j)/d_{ij}$

是两个不同个体之间的位移方向，A_i、B_i 和 k 设为常数以用于控制相互作用的强度，而 $g(x)=\begin{cases}x, & x>0\\ 0, & x\leqslant 0\end{cases}$ 表示了个体之间的身体接触情况；第二项则表示了防止个体滑倒的摩擦力情况，它是跟运动的切向方向 $\boldsymbol{t}_{ij}$ 和速度差 Δv_{ji}^{t} 相关。



(a) 个体之间的作用　　(b) 个体与环境之间的作用

图 9.15　社会力模型

H_i 项反映了环境中的障碍物或者活动范围边界对个体的影响，也称为环境力。该项对应的力学公式为

$$f_{iw}=\{A_i\exp[(r_i-d_{iw})/B_i]+kg(r_i-d_{iw})\}\boldsymbol{n}_{iw}-kg(r_i-d_{iw})(\boldsymbol{v}_i\cdot\boldsymbol{t}_{iw})\boldsymbol{t}_{iw} \tag{9.12}$$

其中，d_{iw} 为个体到环境 w 之间的距离(如图 9.15(b)所示)。

通过公式(9.10)就可以准确地描述社会力对人群中个体的影响，而且通过参数的设置来调整恐慌或非恐慌等不同状态下的实际运动行为。最终的群体动画则是通过求解公式(9.10)所定义的微分方程来获得。总体而言，社会力模型将行人个体抽象为力学中的质点，借助牛顿力学来对物理和社会心理学影响进行建模，很好地展现出人群聚集的群体行为动画效果。

9.6 小　　结

计算机动画是计算机图形学一个非常重要的应用领域。动画角色可以通过几何建模辅助完成，并通过绘制技术进一步生成连续的图像序列。基于图形建模和绘制技术的关键帧插值是计算机动画的重要问题之一。此外，物理模拟和运动捕捉的方式能够提供更多样、更便捷、更真实的角色动作生成方式，为计算机动画制作提供了新的解决方案。

群体动画作为影视特效的重要内容，受到越来越多的关注。但是目前的方法在场面表现力、角色逼真度等方面还存在很多问题，尤其是场景宏大、角色众多的场面，这也是当前计算机动画研究的热点问题。

思考题

9.1　人眼觉察到动画的基本原理是什么？

9.2 计算机出现之前的传统动画制作方式有哪些？

9.3 动画制作流程中包含哪些重要步骤？各自的作用是什么？

9.4 采用粒子系统生成动画的基本原理是什么？

9.5 运动捕捉的方式有哪些？各自的优势和不足是什么？

9.6 群体动画制作时的基本原则是什么？

第 10 章 基于 GPU 的图形计算

计算机图形学的很多算法中包含大量的数据并行处理任务，可以采用图形处理器(Graphics Processing Unit，GPU)加速计算。GPU，又称为视觉处理器、图形显示芯片等，是一种专门在个人计算机、工作站和一些移动设备(如智能手机等)上进行图形和图像相关运算工作的微处理器。可以说，现代计算机图形学是和 GPU 的发展紧密结合在一起的。

本章简要介绍 GPU 的基础知识，包括 GPU 的结构、并行计算特点、数值计算加速方式。此外，针对计算机图形学中典型的建模、绘制的算法以及计算摄像的算法，简单介绍基于 GPU 的加速方式。

10.1 GPU 简 介

一般来说，GPU 可以看作是由数以千计的更小、更高效的核心组成的大规模并行计算架构，是专为同时处理多重任务而设计的。GPU 与传统的 CPU 在组成结构、计算模式等方面存在着许多不同。

10.1.1 组成结构

GPU 的组成可以从硬件结构和软件结构两方面分别描述。从硬件结构来讲，GPU 的组成包括 SP、SM 和 Warp 三个不同层次的结构单元。SP(Streaming Processor)，也就是流式处理单元，又称为流处理器。SP 是进行指令和任务处理的最基本单元。所有 GPU 程序都是在 SP 上处理，而 GPU 对数据的并行处理，实质上就是很多个 SP 同时做处理。SM(Streaming Multiprocessor)，也就是流式多处理核，又称为流处理器簇。SM 是由多个 SP，再加上内存、寄存器等存储单元构成，对数据做共享和并行处理。SM 就类似于 CPU 的核，而存储单元又有全局的、常量的、共享的等类型。更多的 SM 就意味着 GPU 可以在同一时刻处理更多的任务和数据。Warp，也就是线程包，是 GPU 执行程序时的调度单位。同在一个 Warp 的线程，面向不同数据资源执行相同的指令。简单而言，一个 GPU 是由多个 SM 组成的阵列，而每个 SM 又是由多个 SP 组成的。

GPU 在运行程序时的数据主要从 GPU 负责管理的内存中进行读取，包括全局内存、共享内存、局部内存、寄存器等。全局内存是从 GPU 核心分离出来的一个存储硬件，占据了绝大部分的显存，是 GPU 上存储硬件的最大组成部分，也是 GPU 上 I/O 最慢的内存操作。显卡一般通过 GDDR(Graphic Double Data Rate)接口进行访问。共享内存是

SM 内的存储单元，读写速度非常快，但是只供每个 SM 内部访问，也就是 GPU 为每个 SM 提供唯一的共享内存。局部内存，是将数据保存于显存，读写速度也很慢。寄存器则是可以直接被 GPU 处理器访问的存储单元，是 GPU 内部最快的内存。寄存器的访问速度约是共享内存的 10 倍。

从软件结构来讲，GPU 在程序运行时，主要包括线程(Thread)、块(Block)、网格(Grid)三个级别的资源组织。最小的逻辑单位是一个线程，是并行程序的基本构建块。最小的硬件执行单位是线程包。若干个(典型值是 128～512 个)线程组成一个块，块被加载到 SM 上运行，多个块组成整体的网格。但是，CPU 与 GPU 对线程的支持方式是不同的。例如，由于 CPU 的每个核只有少量寄存器，导致在进行任务与任务之间上下文切换时，需要先将寄存器的数据转移到 RAM 中，等到重新执行这个任务时，再从 RAM 转移回来。而 GPU 则拥有多个寄存器，那么很容易地对当前寄存器内容进行换进换出。

虽然 GPU 和 CPU 都是处理器，但是如图 10.1 所示，两者在设计架构上有着根本的不同。一般来说，CPU 的设计是用来执行少量比较复杂的任务，而 GPU 的设计则是用来执行大量比较简单的例子。因此，GPU 的 40%是 ALU(算术逻辑单元)，但 Cache(缓冲存储器)的数量很少，只有非常简单的控制逻辑。而 CPU 的微架构是按照兼顾“指令并行执行”和“数据并行运算”的思路而设计。CPU 的大部分晶体管主要用于构建控制电路和 Cache。CPU 的 5%是 ALU，控制电路设计更加复杂。CPU 不仅被 Cache 占据了大量空间，而且还有复杂的控制逻辑和诸多优化电路。然而，CPU 有强大的 ALU。它可以在很少的时钟周期内完成算术计算。因此，对于单个线程的处理能力来说，CPU 更强。但是，GPU 采用了数量众多的计算单元和超长的流水线，整体计算能力更强。

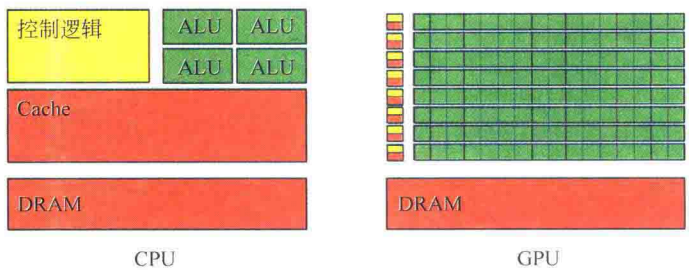


图 10.1 CPU 和 GPU 的结构比较

10.1.2 并行处理

并行处理的类型包括基于任务的并行处理、基于数据的并行处理等不同类型。基于任务的并行处理是将不同的任务无冲突地同时分配给若干个处理器，其关键是各个处理器的任务处理要互相独立。这是一种粗粒度的并行处理模式。基于数据的并行处理是将待处理的数据做分配，不同的处理器分别处理各个数据块。后者更关注于数据及其所需的交换，而不是待执行的任务。

GPU 可以利用众多的计算核心，实现两种类型的并行处理。在同一时刻，GPU 能够并行处理多个数据，解决极高的运算密度、并发线程数量和频繁地存储器访问等数据访问和计算问题。因此，GPU 适合处理计算密集型任务和易于并行的任务。

所谓计算密集型的程序，是指大部分运行时间花在了寄存器的运算。利用 GPU 的并行计算模式，可以高速、频繁地访问寄存器，因此适合于计算密集型程序。从另外一个角度讲，GPU 其实是一种 SIMD(Single Instruction Multiple Data)架构，具有成百上千个核，而每一个核在同一时间能做同样的事情，因此适合执行那些易于并行的任务。

GPU 的并行处理能力最早是为了满足图形学算法任务的需求。典型的算法如第 5 章中的光线跟踪算法，非常适合 GPU 并行地处理大量光线的跟踪和绘制。后来，随着通用图形处理器(General Purpose GPU，GPGPU)概念的提出和发展，GPU 的使用越来越广泛，朝着“处理单元功能不再是固定”的这一方向迅速发展。

通常来讲，GPU 比 CPU 的计算能力提升更为迅速，而且功耗的增加又相对平缓。很长时间以来，CPU 性能的发展遵循摩尔定律，也就是说晶体管的密度每过 18 个月就会翻一番，而其性能提升一倍。但当今的半导体技术已经接近了诸多物理极限。晶体管的大小不可能小于一个分子的大小。事实上，在现实世界的应用中，五个分子大小已经是半导体所能实现的最小尺寸。因此，目前 CPU 的发展进入了缓慢期。

在 2008 年到 2017 年的十年时间里，CPU 的计算能力提升不到 2 倍，而 GPU 的计算能力提升 7 倍。尤其是在 2009 年，GPU 的性能突破每秒 1 万亿次的大关后，GPU 的计算能力就远远超出了 CPU。此外，一个 CPU 的功耗通常被认为是其时钟频率的三次方，这也使得 CPU 设计也面临电子设备散热极限。因此，利用 GPU 进行并行处理，是进一步提升计算机和服务器计算能力、实现更高性能计算的首选途径。

GPU 的并行处理需要提供一些易用的编程接口，才能方便用户开展并行计算。2007 年，NVIDIA 公司发布了面向 GPU 的统一计算架构(Compute Unified Device Architecture，CUDA)，为直接进行 GPU 编程提供了可能。一般来讲，CUDA 是 C 语言的一种扩展，而 C 语言又是使用最广泛的一种高级编程语言。因此，通过 C 语言来进行 GPU 代码编程，能够更有利于用户采用 GPU 并行处理来提高计算性能。可以说，CUDA 和 GPU 正在改变着高性能计算的发展模式。除了 CUDA 以外，Apple 公司的 OpenCL(Open Computing Language)也是一种开放的、免费的、支持 GPU 编程的框架。

10.1.3　GPU 的发展

GPU 的发展是伴随着芯片图形处理能力的提升而不断更新换代。目前，NVIDIA 和 AMD 是最主要的显卡制造厂商，也是 GPU 硬件和计算框架发展的最有力推动者。从 20 世纪 90 年代针对计算机游戏的第一款显卡面世，到如今虚拟现实、自动驾驶、人工智能等领域显卡的广泛使用，GPU 经历了飞速的发展。图 10.2 展示了从第一代到第五代的 GPU 显卡。

第一代 GPU 将图形绘制中部分片元(Fragment)的操作放在 GPU 上执行，如简单的纹理映射、z-缓存处理。几何(Geometry)元素的操作还是在 CPU 上。典型的是 1996 年

图 10.2 GPU 显卡的发展

的 3Dfx Voodoo 显卡所具备的 GPU，这也是第一款 3D 游戏视频卡。

第二代 GPU 将几何元素的操作也转移到 GPU 上去执行，同时能够执行更多种类型的纹理映射，如凹凸映射、环境光映射。在 CPU 到 GPU 的数据传输方面，也使用特殊的图形加速接口(AGP)取代通用的 PCI 进行传输。这一代的典型代表是 NVIDIA 在 1998 年推出的 GeForce 7800 显卡所具备的 GPU。

从第三代 GPU 开始，显卡制造厂商允许对 GPU 进行编程。例如 NVIDIA 在 2001 年推出的 GeForce 3 显卡，可以通过顶点着色器的编程进行几何操作。此外，支持体纹理、多重采样等更多类型的纹理操作。

第四代 GPU 可以对顶点着色器(Vertex shader)和片元着色器(Fragment shader)进行编程，同时可以直接访问纹理内存，进行更高效的 GPU 内部数据读取。

第五代 GPU 是随着通用图形处理器的出现而发展，能够在使用几何着色器时，对几何元素进行更方便地操作。

10.2 数值计算

GPU 凭借其强大的并行运算能力，不仅在传统的图形图像处理领域发挥着重要作用，也越来越多地被应用于科学计算领域。

10.2.1 计算模式

利用 GPU 开展并行计算时，不需要将整个应用程序移植到 GPU。通常是将应用程序中涉及的性能瓶颈部分，同时也是计算密集型的子任务，转入 GPU 进行并行处理。其余的任务依旧可以按照原样运行在 CPU 上。

GPU 程序通常以内核函数形式运行。所谓内核函数，是指一个只能在 GPU 上执行而不能直接在 CPU 上执行的函数。在 GPU 执行计算任务时，相应的内核被唤醒，而且任务被分解为线程。以图 10.3 所示的一段循环体为例，它将两个数组 b 和 c 中相同下标的元素进行相乘运算，然后结果保存在数组 a 中。

```
void sum_func (void)
{
    int i;
    for (i=0; i<256; i++)
        a[i] = b[i]*c[i];
}
```

图 10.3 CPU 运行的循环体函数

CPU 中的串行代码需要执行 256 次 for 循环，而 GPU 则可以通过创建内核函数，将代码直接转换成 256 个线程来并行执行。其中，每个线程都执行 a[i]＝b[i]＊c[i]的操作。从概念上看，GPU 的内核函数和图 10.3 中的循环体是一样的。图 10.4 所示的代码是 CUDA 编程时所对应的内核函数，其中没有了循环体，取而代之的是被每个线程执行的操作。

```
_global_ void sum_kernel_func (int * const a, const int * const b, const in * const c)
{
    a[i]=b[i]*c[i];
}
```

图 10.4　GPU 运行的内核函数

一般来讲，线程在 GPU 内以网格和块的形式进行组织：线程组成块，块组成网格。然后，每个网格对应一个内核。因此，如图 10.5 所示，线程作为基本的执行单元可以看作一种三维数组的形式，而内核执行计算时的线程数就是由网格数和块数所决定的。

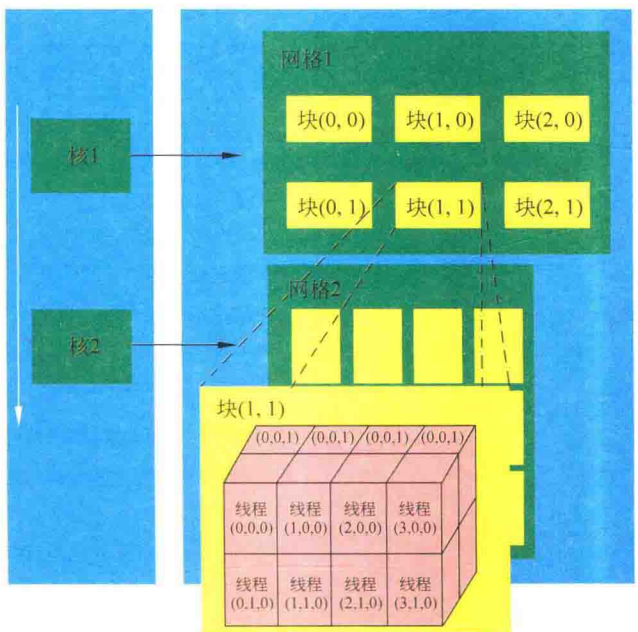


图 10.5　并行计算时的线程组织模式

在 SM 内部，块分解为执行共同指令的线程包。因此，一个线程包中的所有线程并行地执行相同的指令，如图 10.4 所示的函数。一个线程包在执行时需要占用一个 SM。这样，多个线程包就需要轮流进入 SM 运行，也就是说一个 SM 可以同时拥有多个块，但需要按顺序依次执行。具体的调度则由 SM 的硬件 Warp 调度器负责完成。

从 GPU 的硬件结构来看，SM 是 GPU 的心脏，通过以 SP 为单位进行资源调度和处理。从 GPU 的软件结构来看，块是基本调度单位，实际是执行相同指令的线程包。一个

块只会由一个 SM 调度。在并行计算时，通过设定块的属性，告诉 GPU 硬件，需要占用多少个线程、线程怎么组织，以此完成相应的计算任务。

10.2.2 通用数值计算

接下来，通过两个具体的例子来展示 GPU 数值计算的方式，并与 CPU 计算效率进行比较。这两个例子在相同的计算机上进行测试，使用的处理器型号是 Intel i5-2400，主频 3.1GHz，显卡型号是 NVIDIA Quadro2000。

第一个例子是并行地将数组中存储的数值递增 1，也就是执行 x[i]=x[i]+1 的操作。传统的 CPU 程序是遍历该数组，然后将数组的元素依次增加 1。GPU 程序则会将数组元素赋予不同的线程，通过线程的并行来执行加 1 操作，完成数组元素递增。

图 10.6 展示了用 C 语言编写的 CPU 和 GPU 程序来实现数组递增操作。虽然两个程序在执行的运算上很相似，但数据组织形式上有很大的不同。对于包含 34 603 008 个元素的数组，CPU 执行递增加 1 的运算需要 156ms，而 GPU 只需要 49ms。

CPU program

```
void inc_cpu (int *a, int N)
{
    int idx;
    for (idx = 0; idx < N; idx++)
        a[idx] = a[idx] + 1;
}
int main()
{
    ...
    inc_cpu (a, N);
}
```

CUDA program

```
_global_void inc_gpu (int *a, int N)
{
    int idx = blockIdx.x * blockDim.x
            + threadIdx.x;
    if (idx < N)
        a[idx] = a[idx]+1;
}
int main()
{
    ...
    dim3 dimBlock (blocksize);
    dim3 dimGrid (ceil(N/(float)
 blocksize);
    inc_gpu<<dimGrid, dimBlock>>(a,N);
}
```

图 10.6 CPU 和 GPU 上分别实现的数组递增操作的代码

第二个例子是并行地将两个矩阵 **A** 和 **B** 进行相加运算，也就是执行 **A**[i][j]+**B**[i][j]的操作。如图 10.7 所示的代码，传统的 CPU 程序是依次遍历矩阵行和列以获得矩阵元素，然后将对应的元素执行加法计算。而 GPU 程序则将矩阵元素赋予不同的线程，并行地进行元素相加操作。对于两个 5120×5120 的矩阵相加，CPU 程序运行时间约为 203ms，而 GPU 程序运行时只需 70ms。

```
CPU program
void add_matrix_cpu (float *a,
    float *b, float *c, int N)
{
    int i, j, index;
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            indx = I + j * N;
            c[index] = a[index]
                +b [index];
        }
    }
}
int main()
{
    …
    add_matrix (a, b, c, N);
}
```

```
CUDA program
_global_void add_matrix_gpu (
          float *a, float *b, float *c,
           int N)
{
     int i = blockIdx.x * blockDim.x
          + threadIdx.x;
     int j = blockIdx.y * blockDim.y
          + threadIdx.y;
     int index = i + j * N;
     if (i < N && j < N)
          c[index] = a[index]
                   + b[index];
}
int main()
{
     dim3 dimBlock(blocksize,
          blocksize);
     dim3 dimGrid (N/dimBlock.x,
          N/dimBlock.y);
    add_matrix_gpu << dimGrid,
        dimBlock >> (a,b,c,N);
}
```

图 10.7　CPU 和 GPU 上分别实现的矩阵加法的代码

10.3　GPU 快速建模

几何建模是计算机图形学的重要内容。充分利用 GPU 的并行计算能力，是有助于提高建模效率的。第 3 章介绍了各种不同的几何建模方法。接下来，以自由曲线/曲面建模中的 NURBS 方法和三维重建中的泊松方法为例，介绍如何通过 GPU 实现更高效的几何建模。

10.3.1　GPU 加速的 NURBS 建模

根据第 3 章中公式(3.12)的定义，NURBS 曲面本质上是一种有理 B 样条曲面。而其计算的关键在于 B 样条函数值的求解，这也是通过 GPU 提高 NURBS 建模效率的主要手段。事实上，使用 NURBS 进行建模时，所涉及的计算可以分为两种类型：正向计算和逆向计算。正向计算，是通过已知的参数值 s 和 t，求解 NURBS 曲面上对应点的三维坐标(x,y,z)，以及该点处的切向、法向等高阶导数相关的几何量。逆向计算，则是给定

NURBS 曲面上某一点的三维坐标，计算对应的参数值。这两类计算也成为使用 GPU 加速 NURBS 建模的核心问题。

在正向计算时，由于参数值是已知的，同时根据 B 样条函数的局部支撑性可知，通过 GPU 进行计算时就归结为确定参数值所在的节点区间，以及该节点区间所对应的非零 B 样条基函数集合。在此基础上，进一步借助 de Boor 递归算法直接计算 NURBS 曲面上对应点的三维坐标。这里，如图 10.8 所示，定义 NURBS 曲面时所需要的两个方向（对应于参数值 s 和 t 取值的两个维度）上的节点向量、控制顶点、权重因子等，都可以借助一维或者二维纹理的形式进行存储，并能通过片元着色器实现 GPU 编程。

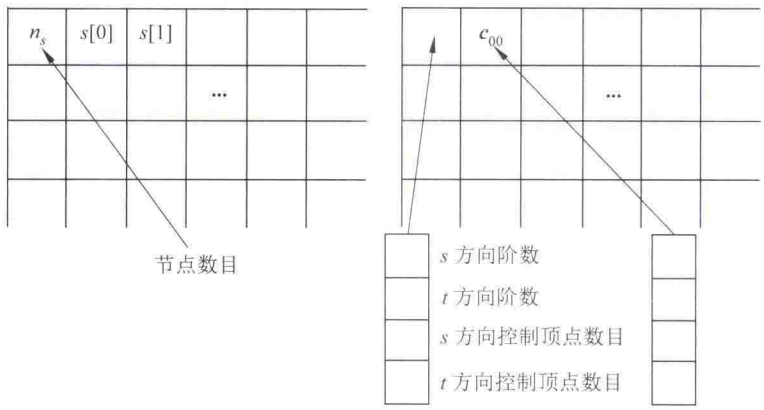


图 10.8 纹理内存中的 NURBS 节点向量、控制顶点和权重因子

以 s 方向为例，节点向量存储于一维纹理内存。首先，给定参数值 s，通常采用 GPU 上并行的二分查找方法，快速确定其所在的区间，记为 $s_0 \in [s_i, s_{i+1})$。然后，根据 k 次（$k+1$ 阶）B 样条基函数的局部支撑性，确定该区间上所对应的至多 $k+1$ 个取值非零的 B 样条基函数，记为 $\{N_{i-k}^k(s), N_{i-k+1}^k(s), \cdots, N_i^k(s)\}$，以及相应的控制顶点，记为 $\{\boldsymbol{c}_{i-k}, \boldsymbol{c}_{i-k+1}, \cdots, \boldsymbol{c}_i\}$。最后，通过 GPU 递归来实现 de Boor 算法，计算该参数值所对应的三维坐标。对于高阶导数，则可以利用第 3 章中公式(3.10)所示的 de Boor-Cox 公式，将求导运算转化为低阶 B 样条基函数的计算，同样也可以实现 GPU 上的快速计算。

在逆向计算时，是通过 NURBS 曲面上给定点的三维坐标，反求相应的 s 和 t 参数值。由于 B 样条函数的非线性，逆向计算往往无法得到精确解，而通常是在给定的误差范围内，计算最优的数值解，使其对应的 NURBS 曲面上的点和给定点之间的距离最小。为此，首先将参数域进行划分，得到不同尺寸大小的格子，并以此构造 NURBS 曲面上的轴对齐包围盒（Axis-aligned bounding boxes，AABB）。这些包围盒的顶点恰好对应于参数域划分时的格子顶点，而它们的参数值是确定的。AABB 是可以通过 GPU 并行方式来快速构造的。然后，给定曲面上的一点，通过连接原点和该点的直线，判断其所在的包围盒以及相应的截面，确定距离最近的包围盒顶点。其中，直线和包围盒求交可通过 GPU 上直线和平面求交快速计算。最后，根据包围盒顶点所对应的参数域格子顶点，确定相应的参数值。这里，AABB 尺寸大小是可以根据预设的误差来设计，从而得到指定误差范围内的数值解。

10.3.2　GPU 加速的泊松三维重建

通过泊松方法对点云数据做三维重建，是几何建模中的典型方法。泊松方法是构建三维点云对应的隐函数，并通过求解泊松方程来恢复点云的三维网格形状。因此，泊松方程的求解是影响三维重建计算效率最重要的环节。

传统泊松重建算法在设计时主要针对 CPU 的架构和计算模式，通过一个隐函数 $F(x,y,z)$ 来计算三维指示函数，并且通过提取该指示函数的等值面来获取重建的表面。一般来讲，该算法整体运行时间较长，无法达到实时重建。

事实上，泊松重建算法有两个关键的步骤：构造三维点云的八叉树和求解泊松方程，而这两个步骤恰巧可以通过 GPU 的并行计算来加速处理。其基本思想是利用 GPU 数据并行的特点，设计特殊的八叉树，并将其与泊松方程的计算相结合，进而实现更快速的泊松重建。

首先，根据输入的点云数据，利用 GPU 多线程实时构建八叉树。输入的点云记为 $Q=\{q_i \mid i=1,2,\cdots,N\}$，需要构建最大深度为 D 的八叉树。这里，采用层次构建策略，每层节点同时进行构建，依次得到相应的顶点数组、边数组、面数组和节点数组，并记录在 GPU 能直接访问的内存中。这里最重要的操作是对节点信息的快速计算和查询。根据 GPU 的数据访问特点，采用 32 位浮点数序列来表示从根节点到叶节点的所有节点，记为 $\{o_i\}=x_1y_1z_1x_2y_2z_2\cdots x_Dy_Dz_D$。同时，根据节点的位置和深度，建立相邻节点查找表，以便于 GPU 计算时的快速查询。

然后，将泊松方程求解过程与八叉树相结合，在 GPU 上进行快速计算，并提取网格表示的重建表面。为了方便计算八叉树节点处的隐函数 $F(x,y,z)$，采用如下形式的箱函数作为基函数进行表示：

$$F(x,y,z) = f_{o.x,o.w}(x) f_{o.y,o.w}(y) f_{o.z,o.w}(z) \tag{10.1}$$

这里，o 表示节点位置，w 代表箱函数的三个变量在各自定义域的取值范围，即满足 $\{x,y,z\}\in[-w/2,w/2]$。那么，通过对八叉树每个节点处基函数值的线性组合，就可以得到和点云形状相对应的三维指示函数，并将泊松方程转化为相应的 Laplaican 方程。

接下来，采用 GPU 上的共轭梯度法计算 Laplacian 方程的数值解，以及相应的等值面。最后，采用 3.4 节中的 Marching cube 方法抽取网格表面，得到重建的三维网格模型。通过 GPU 实现的泊松重建相比于传统的 CPU 重建，能够提高两个数量级的计算速度。例如，对于 50 万个顶点规模的点云，能够达到以 5 帧/秒的速度进行重建结果的快速计算和显示，因此也支持以交互的方式进行重建（如图 10.9 所示）。

(a) 三维点云　　(b) 用户交互　　(c) 泊松重建结果

图 10.9　支持交互的 GPU 快速泊松重建（图片来自[124]）

10.4 GPU 快速绘制

在第 5 章真实感绘制中，介绍了光线跟踪、辐射度等全局技术。虽然这些绘制方法能够生成高质量的绘制效果，但算法复杂度相对较高，需要的计算量较大。这也使得它们往往只能进行离线绘制，很难达到实时绘制和显示的程度。然而，无论是光线跟踪方法，还是辐射度方法，都包含大量的重复计算，非常适合于 GPU 的并行处理模式。因此，通过 GPU 加速光线跟踪、辐射度等方法，是实现实时全局绘制的有效途径。接下来，简单介绍如何通过 GPU 并行计算来提高光线跟踪和辐射度绘制的效率。

10.4.1 GPU 加速的光线跟踪绘制

事实上，光线跟踪方法需要大量的光线和场景求交运算。这使得复杂场景的绘制速度非常慢，难以达到实时绘制的效果。然而，场景中的绝大部分光线是能够同时跟踪的，而且相邻的光线在场景中传播时具有相似的光线传播路径。这样就可以充分利用 GPU 的并行处理能力，加速光线跟踪时的计算效率。

基于 GPU 的光线跟踪方法主要是利用 GPU 的线程并行，构建面向 GPU 的数据处理方式和数据结构，以提高光线跟踪过程中几何元素遍历、求交、着色等操作的效率。2002 年，美国 University of Illinois 的科研人员开发了一套光线跟踪系统 Ray Engine，最早将 GPU 用于光线跟踪。该系统是把光线和组成场景三维模型的三角面片的求交计算转移到 GPU 的着色器，以提高绘制的效率。

事实上，K-D 树被证明是加速光线跟踪的最有效的数据结构，尤其是在绘制静态场景时。K-D 树是一种二叉树，采用坐标轴对齐的平面对三维空间进行划分，形成不同尺度的格子单元作为叶节点。传统基于堆栈的 K-D 树递归构造策略并不适合于 GPU 并行模式。这里，根据光线跟踪的算法特点，采用光线与每个格子平面的相交情况，确定需要进行求交运算的网格面片集合，并依次将光线穿过的那些格子压入堆栈。这样，就避免了对所有格子的堆栈操作，大大提高了计算效率。这也是利用 GPU 提高光线跟踪绘制的重要手段。

基于早期 ATI Radeon X1900 XTX 显卡进行测试，通过上述 GPU 加速手段实现的光线跟踪绘算法，普遍能够达到 15 帧/秒以上的绘制速度，而且不会降低真实感程度，如图 10.10 所示。这样就可以对绘制的场景进行交互操作，并实时地显示交互后的场景画面。

10.4.2 GPU 加速的辐射度绘制

对于辐射度绘制，在 5.5 节中详细分析了算法时间的消耗。其中，场景离散化为面片后，不同面片之间形式因子的计算和相应辐射度数值的计算占据约 90% 的时间。因此，通过 GPU 加速辐射度绘制的关键，主要在于对这两个步骤的并行处理。

形式因子是描述场景中两个面片之间辐射能量的传递，因此一般需要消耗 $O(n^2)$ 的计算和存储资源，其中 n 是场景中面片的总数。而辐射度计算则是对所有在可见范围内

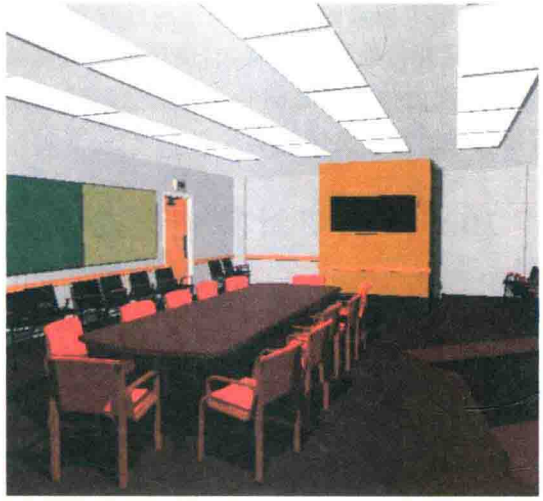


图 10.10　GPU 加速的光线跟踪绘制效果(图片来自[127])

面片传递的能量进行叠加。在 GPU 实现时,可以采用辐射度纹理(Radiosity texture)和残差纹理(Residual texture)两种纹理内存来记录面片上的辐射度数值。这样就可以借助片元着色器来做相应的计算。

辐射度纹理用于保存根据形势因子计算得到的当前传递能量的辐射度值。这里,给定在可见范围内的两个面片后,其对应的形式因子是不依赖于其他面片的。因此可以在着色器上并行地计算形式因子及对应的辐射度数值,同时将相应的结果保存于辐射度纹理内存。事实上,辐射度纹理也对应于当前面片所在的可见范围。残差纹理则用于保存所有面片传递能量的平均值,反映了计算下一次能量传递时所对应的面片。那么,通过不断更新两个纹理内存中的数值,就可以实现对场景画面的迭代绘制。最终的绘制结果是在迭代终止状态下,对辐射度纹理进行映射,以此得到画面中相应的像素颜色。

基于早期的 NVIDIA GeForce FX5900 显卡进行测试,采用上述 GPU 加速手段实现的辐射度方法,在绘制如图 10.11 所示的包含上万个面片的场景时,能够达到 2 帧/秒的

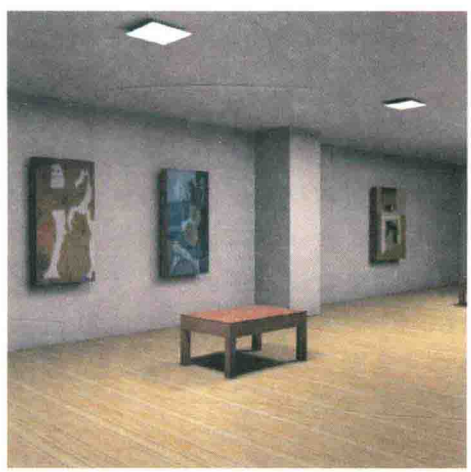


图 10.11　GPU 加速的辐射度绘制效果(图片来自[128])

绘制速度。此外，对于绘制过程中场景的均匀或自适应面片划分、可见性计算时的遮挡判断等，也是能够在 GPU 上进行并行处理的。例如，使用半立方体法计算面片之间的形式因子时，可以将面片连线和半立方体表面之间的几何求交运算，放在顶点着色器上进行计算。这也会进一步提高辐射度绘制的整体效率。

10.5 GPU 计算光谱成像

在第 8 章计算成像中，介绍了 DCCHI 等通过编/解码计算的方式重建高光谱图像的方法。由于光谱图像普遍数据量大，导致计算重建过程中的计算量大。此时传统的 CPU 串行计算已无法满足实时重建任务的需求，因此需要借助 GPU 进行并行加速，在进行高质量重建时也能提高重建速度。

通过对公式(8.6)～(8.11)所示的计算过程进行分析可以发现，影响计算速度的主要因素包括系统响应矩阵 $\boldsymbol{H}$ 和差分矩阵 $\boldsymbol{D}$ 的计算。因此，在并行加速时需要着重对这两个步骤进行 GPU 上的并行优化处理。

首先对系统响应矩阵 $\boldsymbol{H}$ 的计算进行并行优化。对于全色相机分支的前向响应矩阵 $\boldsymbol{H}_u$，由于其在高光谱图像上的作用主要表现为公式(8.8)的积分过程，因而只需要开辟与 $\boldsymbol{G}_u$ 相同大小的线程并共享内存空间，然后进行逐谱带叠加即可。而在对 CASSI 系统分支的前向响应矩阵 $\boldsymbol{H}_c$ 进行并行优化时，就需要考虑公式(8.6)中色散移位过程，也就是 m-λ 的影响。通常情况下，串行方法是先将光谱数据进行移位和补零操作，然后再完成后续操作的过程。这种方法原理简单，但在 GPU 中进行数据移位需要开辟额外的内存空间。事实上，对于二维压缩采样的每一行，上面的数据都是由相同波段的光谱图像叠加而成，所对应的叠加起始波段和结束波段都是固定的。根据这个特点，就可以建立基于查找表的 CASSI 系统响应模型优化。这里定义一个大小为 $2\times(N+\Lambda-1)$ 的查找表，用来标记不同谱带经过色散棱镜后偏移的距离。该查找表可定义为：

$$LUT(1,x)=\begin{cases}1, & x\leqslant N\\ x-\Lambda+1 & \text{其他}\end{cases},\quad LUT(2,x)=\begin{cases}x, & x\leqslant\Lambda\\ \Lambda & \text{其他}\end{cases} \tag{10.2}$$

那么，利用此查找表，公式(8.6)中的 CASSI 成像过程可以重新改写为：

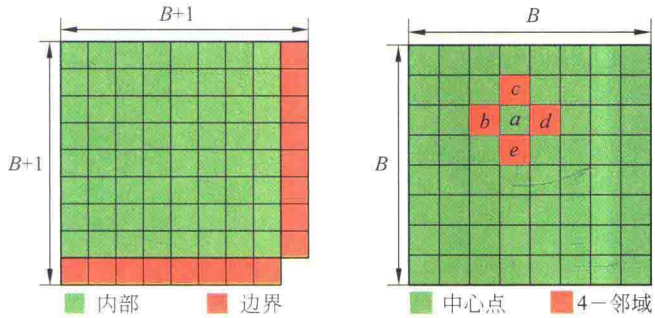
$$g_c(m,n)=\sum_{i=LUT(1,m)}^{LUT(2,m)}H_c(m-i,n,\lambda)f(m-i,n,\lambda) \tag{10.3}$$

这样就不再需要开辟额外的内存空间，而且并行性能得到明显提升。

接下来对差分矩阵 $\boldsymbol{D}$ 的计算进行优化。差分计算需要频繁对数据进行读取操作，因此需要利用共享内存。对于前向差分 $\boldsymbol{DF}$，使用图 10.12(a)所示的共享内存结构进行数据读写优化，其中 B 表示分配的线程块的边长大小。由于差分操作包含边界处理，所以共享内存大小为 $(B+1)\times(B+1)$。具体来讲，首先将图像数据从 GPU 的全局内存中先预加载到共享内存。然后，在进行差分操作时，数据就从共享内存中直接读取，并将差分结果存至全局内存。而对于反向差分，则可以直接在全局内存中进行读写操作即可。

在使用交替方向乘子法求解公式(8.11)所示的光谱图像重建方程时，进一步采用如

图 10.12(b)所示的一种中心共享内存优化模式(Center Shared Memory,CSM)。具体来说,在使用交替方向乘子法计算$\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}(a)$时,中心点 a 的数据从共享内存中读取,而其四邻域点 b、c、d、e 则从全局内存中读取。CSM 模式将共享内存与全局内存相结合,不仅利用共享内存数据存取快的优点,而且避免了内存访问冲突,在 GPU 上的并行计算效率能够得到显著提高。



(a) 前向差分优化　　(b) CSM 模式

图 10.12　共享内存优化结构

图 10.13 展示了高光谱图像在波长为 600nm 时的重建结果。从这个例子可以看到,使用 GPU 并行重建能够达到与 CPU 串行相同的重建质量,但是计算效率却能够提升 100 倍。

(a) 参考图像　　(b) CPU 上的重建　　(c) GPU 上的重建

图 10.13　CPU 和 GPU 上的光谱图像重建结果

10.6　小　　结

GPU 的出现伴随着计算机图形学技术的发展而产生,同时也深刻影响了计算机图形学的发展。从某种意义上讲,GPU 可以进行几乎全部与计算机图形学有关的数据运算,为图形建模、绘制等提供了强大的计算能力。

事实上,图形学之外的领域也开始注意到 GPU 与众不同的计算能力,把 GPU 用于通用计算(GPGPU),以解决更多的计算密集型问题,尤其是在深度学习被广泛使用的人工智能、大数据处理等领域,GPU 正发挥着巨大的作用。

思考题

10.1 GPU 的硬件结构和软件结构有什么特点？它和 CPU 的区别有哪些？

10.2 如何使用 GPU 进行并行计算？

10.3 举例说明 GPU 如何用于提高几何建模的效率。

10.4 举例说明 GPU 如何用于提高真实感绘制的效率。

10.5 举例说明 GPU 如何用于提高计算光谱成像的效率。

参考文献

[1] Sutherland I. Sketchpad, a man-machine graphical communication system. PhD thesis, MIT, 1963.

[2] 孙家广,胡事民. 计算机图形学基础教程. 北京:清华大学出版社,2005.

[3] 彭群生,金小刚,万华根,冯洁青. 计算机图形学应用基础. 北京:科学出版社,2009.

[4] Hill Jr. F. R., Kelley S. M. Computer graphics using OpenGL (3rd Edition). Englewood: Prentice Hall, 2006.

[5] Kessenich J., Sellers G., Shreiner D. OpenGL 编程指南. 王锐,等译. 北京:机械工业出版社,2017.

[6] Hill Jr. F. R., Kelley S. M. Computer graphics using OpenGL (3rd Edition). Englewood: Prentice Hall, 2006.

[7] Fernando R., Kilgard M. The CG tutorial: The definitive guide to programmable real-time graphics. Boston: Addison-Wesley, 2003.

[8] 王国瑾,汪国昭,郑建民. 计算机辅助几何设计. 北京:高等教育出版社,2001.

[9] Hartley R., Zisserman A. Multiple view geometry in computer vision. Cambridge: Cambridge University Press, 2004.

[10] Snavely N., Simon I., Goesele M., Szeliski R., Seitz M. Scene reconstruction and visualization from community photo collections. Proceedings of the IEEE, vol. 28, no. 8, 1370-1390, 2010.

[11] Dey T. K. Curve and surface reconstruction: algorithms with mathematical analysis. Cambridge: Cambridge University Press, 2007.

[12] Ahn S. J., Yoo J., Lee B. G., Lee J. J. 3D surface reconstruction from scattered data using moving least square method. Proceedings of ICIAP, 719-726, 2005.

[13] Kazhdan M., Bolitho M., Hoppe H. Poisson surface reconstruction. Proceedings of SGP, 61-70, 2006.

[14] Izadi S., Kim D., Hilliges O., Molyneaux D., Newcombe R., Kohli P., Shotton J., Hodges S., Freeman D., Davison A., Fitzgibbon A. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. Proceedings of UIST, 559-568, 2011.

[15] Smith A. Plants, Fractals, and Formal Languages. Proceedings of SIGGRAPH, 1-10, 1984.

[16] Stroud I., Nagy H. Solid modeling and CAD systems. New York: Springer, 2011.

[17] 王国瑾,刘利刚. 几何计算、逼近与处理. 北京:科学出版社,2015.

[18] Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B. Polygon mesh processing. Boca Raton: CRC Press, 2010.

[19] Nealen A., Igarashi T., Sorkine O., Alexa M. Laplacian mesh optimization. GRAPHITE, 2006.

[20] Hoppe H. Progressive mesh. ACM SIGGRAPH, 99-108, 1996.

[21] Sheffer A., Praun E., Rose K. Mesh parameterization methods and their applications. Foundations and Trends in Computer Graphics and Vision, 2(2), 105-171, 2006.

[22] Floater M. Parameterization and smooth approximation of surface triangulations. Computer Aided Geometric Design, 14(3), 231-250, 1997.

[23] Sheffer A, de Sturler E. Parameterization of faceted surfaces for meshing using angle based flattening. Engineering with Computers, 17(3), 326-337, 2001.

[24] Lévy B., Petitjean S., Ray N., Maillot J. Least squares conformal maps for automatic texture atlas generation. ACM SIGGRAPH, 362-371, 2002.

[25] Gotsman C., Gu X. F., Sheffer A. Fundamentals of spherical parameterization for 3D meshes. ACM SIGGRAPH, 358-363, 2003.

[26] Tarini M., Hormann K., Cignoni P., Montani C. PolyCube-Maps. ACM SIGGRAPH, 853-860, 2004.

[27] Kraevoy V., Sheffer A. Cross-parameterization and compatible remeshing of 3D models. ACM SIGGRAPH, 861-869, 2004.

[28] Chen Z. G., Cao J., Wang W. P. Isotropic surface remeshing using constrained centroial Delaunay mesh. Computer Graphics Forum, 31(7), 2077-2085, 2012.

[29] Yan D. M., Lévy B., Liu Y., Sun F., Wang W. P. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. Computer Graphics Forum, 28(5), 1445-1454, 2009.

[30] Gu X. F., Gortler S., Hoppe H. Geometry image. ACM Transactions on Graphics, 21(3), 355-361, 2002.

[31] Sederberg T. W., Parry S. R. Free-form deformation of solid geometric models. Computer Graphics, 20(4), 151-160, 1986.

[32] Ju T., Schaefer S., Warren J. Mean value coordinates for closed triangular meshes. ACM Transactions on Graphics, 24(3), 561-566, 2005.

[33] Sorkine O., Cohen-Or D., Lipman Y., Alexa M., Rossl, Seidel H. P. Laplacian surface editing. Symposium on Geometry Processing, 175-184, 2004.

[34] Summer, R. W., Popovic J. Deformation transfer for triangle meshes. ACM Transactions on Graphics, 22(3), 399-405, 2004.

[35] Lee A. W. F., Dobkin D., Sweldens W., Schroder P. Multiresolution mesh morphing. ACM SIGGRAPH, 343-350, 1999.

[36] Xu D., Zhang H. X., Wang Q., Bao H. J. Poisson shape interpolation. Symposium on Solid and Physical Modeling, 267-274, 2005.

[37] 彭群生,鲍虎军,金小刚. 计算机真实感图形的算法基础. 北京：科学出版社,1999.

[38] Matusik W., Pfister H., Brand M., McMillan L. A data-driven reflectance model. ACM Transactions on Graphics, 22(3), 759-769, 2003.

[39] Akenine-Moller T., Haines E., Hoffman N. Real-time rendering. Boca Raton: CRC Press, 2008.

[40] Lafortune E., Foo S. C., Torrance K., Greenberg D. Non-linear approximation of reflectance functions. Proceedings of SIGGRAPH, 117-126, 1997.

[41] Elmar E, Michael S, Ulf A, Michael W. 实时阴影技术. 王锐,苏敏,译. 北京：清华大学出版社,2013

[42] Annen T., Mertens T., Seidel H. P., Flerackers E., Kautz J. Exponential shadow maps. Proceedings of GI, 155-161, 2008.

[43] Xu K., Ma L. Q., Ren B., Wang R., Hu S. M. Interactive hair rendering and appearance editing under environment lighting. ACM Transactions on Graphics, 30(6), 173: 1-173: 10, 2011.

[44] Chalmers A., Reinhard E., Davis T. Practical parallel rendering. Boca Raton: CRC Press, 2002.

[45] Strothotte T., Schlechtweg. 非真实感图形学——造型、绘制与动画技术. 叶修梓,万华根,张引,译. 北京:电子工业出版社,2004.

[46] Winkenbach G., Salesin D. H. Computer-generated pen-and-ink illustration. ACM SIGGRAPH, 91-100, 1994.

[47] Hertzmann A. Painterly rendering with curved brush strokes of multiple sizes. ACM SIGGRAPH, 453-460, 1998.

[48] Huang H., Fu T. N., Li, C. F. Painterly rendering with content-dependent natural paint strokes. The Visual Computer, 27(9), 861-871, 2011.

[49] Zeng K., Zhao M. T., Xiong C. M., Zhu S. C. From image parsing to painterly rendering. ACM Transactions on Graphics, 29(1), 2: 1-2: 11, 2009.

[50] Zang Y., Huang H., Li C. F. Stroke style analysis for painterly rendering. Journal of Computer Science and Technology, 28(5), 762-775, 2013.

[51] Curtis C. J., Anderson S. E., Seims J. E., Fleischer K. W., Salesin D. H. Computer-generated watercolor. ACM SIGGRAPH, 421-430, 1997.

[52] Salisbury M. P., Wong M. T., Hughes J. F., Salesin D. H. Orientable textures for image-based pen-and-ink illustration. ACM SIGGRAPH, 401-406, 1997.

[53] Hertzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H. Image analogies. ACM SIGGRAPH, 327-340, 2001.

[54] Wang B., Wang W. P., Yang H. P., Sun J. G. Efficient example-based painting and synthesis of 2D directional texture. IEEE Transactions on Visualization and Computer Graphics, 10(3), 266-277, 2004.

[55] Kang H., Lee S. Y., Chui, C. K. Flow-based image abstraction. IEEE Transactions on Visualization and Computer Graphics, 15(1), 62-76, 2008.

[56] Winnemoller H., Olsen S. C., Gooch B. Real-time video abstraction. ACM Transactions on Graphics, 25(3), 1221-1226, 2006.

[57] Zang Y., Huang H., Zhang L. Guided adaptive image smoothing via directional anisotropic structure measurement. IEEE Transactions on Visualization and Computer Graphics, 21(9), 1015-1027, 2015.

[58] Litwinowicz P. Processing images and video for an impressionist effect. ACM SIGGRAPH, 407-414, 1997.

[59] Huang H., Zhang L., Fu T. N. Video painting via motion layer manipulation. Computer Graphics Forum, 29(7), 2055-2064, 2010.

[60] Collomosse J. P., Rowntree D., Hall P. M. Stroke surfaces: temporally coherent artistic animations from video. IEEE Transactions on Visualization and Computer Graphics, 11(5), 540-549, 2005.

[61] Selim A., Elgharib M., Doyle L. Painting style transfer for head portraits using convolutional neural networks. ACM Transactions on Graphics, 35(4), 129: 1-129: 18, 2016.

[62] Gatys L. A., Ecker A. S., Bethge M. Image style transfer using convolutional neural networks. CVPR, 2016.

[63] Zhu J. Y., Park T., Isola P., Efros A. A. Unpaired image-to-image translation using cycle-

consistent adversarial networks. ICCV, 2017.

[64] Gonzalez R. C., Woods R. E. 数字图像处理. 3 版. 阮秋琦,阮宇智,等译. 北京:电子工业出版社,2017.

[65] Wang J., Cohen M. F. Image and video matting: a survey. Foundations and Trends in Computer Graphics and Vision, 3(2), 97-175, 2007.

[66] Smith A. R., Blinn J. F. Blue screen matting. ACM SIGGRAPH, 259-268, 1996.

[67] Chuang Y. Y., Curless B., Salesin D. H., Szeliski R. A Bayesian approach to digital matting. IEEE CVPR, 2001.

[68] Sun J., Jia J. J., Tang C. K., Shum H. Y. Poisson matting. ACM Transactions on Graphics, 23(3), 315-321, 2004.

[69] Guan Y., Chen W., Liang X., Ding Z. A., Peng Q. S. Easy matting - a stroke based approach for continuous image matting. Computer Graphics Forum, 25(3), 567-576, 2006.

[70] Levin A., Lischinski D., Weiss Y. A closed-form solution to natural image matting. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30(2), 228-242, 2008.

[71] Sun J., Li Y., Kang S. B., Shum H. Y. Flash matting. ACM Transactions on Graphics, 25(3), 772-778, 2006.

[72] Chuang Y. Y., Agarwala A., Curless B., Salesin D. H., Szeliski R. Video matting of complex scenes. ACM Transactions on Graphics, 21(3), 243-248, 2002.

[73] Avidan S., Shamir A. Seam carving for content-aware image resizing. ACM Transactions on Graphics, 26(3), 2007.

[74] Zhang G. X., Cheng M. M., Hu S. M., Martin R. R. A shape-preserving approach to image resizing. Computer Graphics Forum, 28(7), 1897-1906, 2009.

[75] Rubinstein M., Shamir A., Avidan S. Improved seam carving for video retargeting. ACM Transactions on Graphics, 27(3), 16: 1-16: 9, 2008.

[76] Pérez P., Gangnet M., Blake A. Poisson image editing. ACM Transactions on Graphics, 22(3), 313-318, 2003.

[77] Farbman Z., Hoffer G., Lipman Y., Cohen-Or D., Lischinski. Coordinates for instant image cloning. ACM Transactions on Graphics, 28(3), 67: 1-67: 9, 2009.

[78] Chen T., Zhu J. Y., Shamir A., Hu S. M. Motion-aware gradient domain video composition. IEEE Transactions on Image Processing, 22(7), 2532-2544, 2013.

[79] Szeliski R. Image alignment and stitching: a tutorial. Foundations and Trends in Computer Graphics and Vision, 2(1), 1-104, 2006.

[80] Zaragoza J., Chin T. J., Brown M. S., Suter D. As-projective-as-possible image stitching with moving DLT. IEEE CVPR, 2339-2346, 2013.

[81] Chang C. H., Sato Y., Chuang Y. Y. Shape-preserving half-projective warps for image stitching. IEEE CVPR, 3254-3261, 2014.

[82] Lin C. C., Pankanti S. U., Ramamurthy N., Aravkin A. Y. Adaptive as-natural-as-possible image stitching. IEEE CVPR, 1155-1163, 2015.

[83] Lin K. M., Liu S. C., Cheong L. F., Zeng B. Seamless video stitching from hand-held camera inputs. Computer Graphics Forum, 35(2), 479-487, 2016.

[84] Huang H., Zang Y., Li C. F. Example-based painting guided by color features. The Visual Computer, 26(6-8), 933-942.

[85] Levin A., Lischinski D., Weiss Y. Colorization using optimization. ACM Transactions on Graphics, 22(3), 689-694, 2004.

[86] Schaefer S., McPhail T., Warren J. Image deformation using moving least squares. ACM Transactions on Graphics, 25(3), 533-540, 2006.

[87] Xu K., Li Y., Ju T., Hu S. M., Liu T. Q. Efficient affinity-based edit propagation using K-D tree. ACM Transactions on Graphics, 28(5), 118: 1-118: 6, 2009.

[88] Matsushita Y., Ofek E., Tang X. O., Shum H. Y. Full-frame video stabilization. Proc. CVPR, 2005.

[89] Liu F., Gleicher M., Jin H. L., Agarwala A. Content-preserving warps for 3D video stabilization. ACM Transactions on Graphics, 28(3), 44: 1-44: 9, 2009.

[90] Goldstein A., Fattal R. Video stabilization using epipolar geometry. ACM Transactions on Graphics, 31(5), 126: 1-126: 10, 2012.

[91] 戴琼海,索津莉,季向阳,等. 计算摄像学: 全光视觉信息的计算采集. 北京: 清华大学出版社,2016.

[92] Renner E. Pinhole photography. Waltham: Focal Press, 2008.

[93] Li X., Gunturk B., Zhang L. Image demosaicing: a systematic survey. Proc. SPIE 6822, 2008.

[94] Gijsenij A., Gevers T., van de Weijer. Computational color constancy: survey and experiments. IEEE Transactions on Image Processing, 20(9), 2475-2489, 2011.

[95] Rougeron G., Péroche B. Color fidelity in computer graphics: a survey. Computer Graphics Forum, 17(1), 3-15, 1998.

[96] Nayar S. K. Computational cameras: approaches, benefits and limits. Columbia University Computer Science Technical Reports, 2011.

[97] Baker S., Nayar S. K. A theory of single-viewpoint catadioptric image formation. International Journal of Computer Vision, 35(2), 175-196, 1999.

[98] Zhou C. Y., Cossairt O., Nayer S. Depth from diffusion. IEEE CVPR, 2010.

[99] Du H., Tong X., Cao X., Lin S. A prism-based system for multispectral video acquisition. IEEE ICCV, 2009.

[100] Ashok A., Neifeld M. Pseudorandom phase masks for superresolution imaging from subpixel shifting. Applied Optics, 46(12), 2256-2268, 2007.

[101] Zhou C. Y., Lin S., Nayer S. K. Coded aperture pairs for depth from defocus and defocus deblurring. International Journal of Computer Vision, 93(1), 53-72, 2011.

[102] Ng R., Levoy M., Brédif M., Duval G., Horowitz M., Hanrahan P. Light field photography with a hand-held plenoptic camera. Stanford University Computer Science Tech Report CSTR 2005-02, 2005.

[103] Gupta M., Agrawal A., Veeraraghavan A., Narasimhan S. G. Flexible voxels for motion-aware videography. ECCV, 100-114, 2010.

[104] Park J., Lee M. H., Grossberg M. D., Nayar S. K. Multispectral imaging using multiplexed illumination. IEEE ICCV, 2007.

[105] Gong Y. Z., Zhang S. Ultrafast 3-D shape measurement with an off-the-shelf DLP projector. Optics Express, 18(19), 19743-19754, 2010.

[106] Perazzi F., Sorkine-Hornung A., Zimmer H., Kaufmann P., Watson S., Gross M. Panoramic video from unstructured camera arrays. Computer Graphics Forum, 34(2), 57-68, 2015.

[107] Cossairt O., Gupta M., Nayar S. K. When does computational imaging improve performance? IEEE Transactions on Image Processing, 22(2), 447-458, 2013.

[108] Arce G. R., Brady D. J., Carin L., Arguello H., Kittle D. S. Compressive coded aperture spectral imaging: An introduction. IEEE Signal Processing Magazine, 31(1): 105-115, 2013.

[109] Wang L. Z., Xiong Z. W., Gao D. H., Shi G. M., Wu F. Dual-camera design for coded aperture snapshot spectral imaging. Applied Optics, 54(4): 848-858, 2015.

[110] 鲍虎军,金小刚,彭群生. 计算机动画的算法基础. 杭州:浙江大学出版社,2000.

[111] 雍俊海. 计算机动画算法与编程基础. 北京:清华大学出版社,2008.

[112] Parent R. Computer animation: algorithms and techniques. Burlington: Morgan Kaufmann, 2001.

[113] Alexa M., Cohen-Or D., Levin D. As-rigid-as-possible shape interpolation. ACM SIGGRAPH, 157-164, 2000.

[114] Levi Z., Gotsman C. Smooth rotation enhanced as-rigid-as-possible mesh animation. IEEE Transactions on Visualization and Computer Graphics, 21(2), 264-277, 2015.

[115] Zhao H. L., Fan R., Wang C. C., Jin X. G., Meng Y. W. Fireworks controller. Computer Animation and Virtual Worlds, 20(2-3), 185-194, 2009.

[116] Wang N. Realistic and fast cloud rendering in computer games. ACM SIGGRAPH, 2003.

[117] Chai J. X., Xiao J., Hodgins J. Vision-based control of 3D facial animation. Eurographics Symposium on Computer Animation, 193-206, 2003.

[118] Xia S. H., Gao L., Lai Y. K. Yuan M. Z., Chai J. X. A survey on human performance capture and animation. Journal of Computer Science and Technology, 32(3), 536-554, 2017.

[119] Reynolds C. W. Flocks, herds and schools: a distributed behavioral model. ACM SIGGRAPH, 25-34, 1987.

[120] Helbing D., Farkas I., Vicsek T. Simulating dynamical features of escape panic. Nature, 407, 487-490, 2000.

[121] Femando R. GPU 精粹. 姚勇,王小琴,译. 北京:人民邮电出版社,2006.

[122] Kanai T. Fragment-based evaluation of non-uniform B-spline surfaces on GPUs. Computer-Aided Design & Applications, 4(1-4), 287-294, 2007.

[123] Krishnamurthy A., Khardekar R., McMains S., Haller K., Elber G. Performing efficient NURBS modeling operations on the GPU. IEEE Transactions on Visualization and Computer Graphics, 15(4), 530-543, 2009.

[124] Zhou K., Gong M., Huang X., Guo B. N. Highly parallel surface reconstruction. Microsoft Research TR, 2008.

[125] Carr N. A., Hall J. D., Hart J. C. The ray engine. Proceedings of Graphics Hardware, 37-46, 2002.

[126] Horn D. R., Sugerman J., Houston M., Hanrahan P. Interactive k-d tree GPU traytracing. Proceedings of the Symposium on Interactive 3D Graphics and Games, 167-174, 2007.

[127] Popov S., Günther J., Seidel H. P., Slusallek P. Stackless KD-tree traversal for high performance GPU ray tracing. Computer Graphics Gorum, 26(3), 415-424, 2007.

[128] Coombe G., Harris M., Lastra A. Radiosity on graphics hardware. Proceedings of Graphics Interface, 161-168, 2004.

图书资源支持

感谢您一直以来对清华版图书的支持和爱护。为了配合本书的使用，本书提供配套的资源，有需求的读者请扫描下方的“书圈”微信公众号二维码，在图书专区下载，也可以拨打电话或发送电子邮件咨询。

如果您在使用本书的过程中遇到了什么问题，或者有相关图书出版计划，也请您发邮件告诉我们，以便我们更好地为您服务。

资源下载、样书申请

书圈

扫一扫，获取最新目录

课程直播

我们的联系方式：

地　　址：北京市海淀区双清路学研大厦 A 座 701

邮　　编：100084

电　　话：010-83470236　010-83470237

资源下载：http://www.tup.com.cn

客服邮箱：2301891038@qq.com

QQ：2301891038（请写明您的单位和姓名）

用微信扫一扫右边的二维码，即可关注清华大学出版社公众号“书圈”。

内容简介

计算机图形学是研究真实或虚拟物体在计算机中的图形表示及交互的一门学科，在计算机辅助设计与制造、计算机动画、计算机游戏、计算机仿真、虚拟现实、科学计算可视化等领域有着广泛的应用。随着数字媒体、虚拟/增强/混合现实、大数据、人工智能、可视分析等技术的发展，计算机图形学的理论和方法不断地充实与更新，已经广泛应用到大量的科学和工程领域，成为现代计算机应用中不可缺少的重要分支。

本书涵盖了计算机图形学的众多基础内容，不仅介绍了几何建模、真实感绘制、计算机动画等传统图形学的内容，还着重阐述了数字几何处理、非真实感绘制、基于图形的影像处理、计算摄像、GPU图形计算等图形学的新进展。其中，不少章节的内容都取自近二十年内计算机图形学领域的高水平学术论文和产业技术。此外，本书还给出了很多重要概念和方法的实例，并提供了一些代表性算法的关键实现步骤的代码。

本书不仅可以作为高等院校计算机相关专业的“计算机图形学”课程教材，也可以作为计算机图形学科研工作者和产业界技术人员的参考用书。

ISBN 978-7-302-55271-0

9 787302 552710 >

定价：59.00元