浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得浙江大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：[signature]　　签字日期：2010 年 6 月 2 日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 浙江大学 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：[signature]　　导师签名：[signature]

签字日期：2010 年 6 月 2 日　　签字日期：2010 年 6 月 2 日

致谢

在论文的撰写过程中，首先要感谢我的导师吴庆标教授，是他的耐心的指导和帮助使得我可以顺利的完成毕业论文。吴老师从论文的开始就不断给予我建议和思路，并且在百忙之中时常抽空跟我探讨论文内容以及进展情况，给予了无微不至的关怀。在研究生阶段，导师对我的指引和培养使得我取得了长足的进步，他广博的知识面和严谨的治学态度使我受益匪浅，在此，我向他表示深深的感谢。

同时感谢在研究生阶段相处的同学和老师，大家共同学习，一起进步，留下了许多难忘的时光。

最后，感谢父母的辛劳把我养大，感谢女朋友的支持，我所完成的论文，都离不开你们对我的支持和关爱。

摘要

医学影像可视化是医学和计算机科学计算可视化研究领域的一个重要分支，也一直是学者研究的热门点。

面绘制和体绘制是医学影像三维重建的两种实现方法。本文主要深入研究医学影像三维重建中面绘制和体绘制的原理和方法，然后由医学图像（CT、MRI）的二维断层序列来重建三维图像并与之交互。

本文在深入分析 Marching Cubes 面绘制方法之后，提出了两种改进其运行效率的方法。一是利用立方体棱边共享等值点的方法，减少重复计算，节约计算成本，提高了运行效率。二是利用现在计算机 CPU 多核的特性，提出多线程并行计算面绘制等值点的方法，大大提高了面绘制的速度。

在体绘制算法的研究中，本文首先介绍了计算机图形学光照明模型，然后分析了光线投射体绘制法的原理。最后介绍了 Shear-Warp 体绘制加速方法。

本文应用 Visual C++和开源图形图像处理包对图像断层序列进行计算机处理，用 OpenGL 技术将结果用三维模型显示，完成重建。本文在阐述算法的同时，提出了相应的改进，最后通过在自己创建的软件平台下，进行试验，得到了对比的结果。

关键词：医学影像三维 面绘制 体绘制 OpenGL Visual C++

ABSTRACT

Medical image visualization is an important branch of Medical and Visualization in Scientific Computing, and it is always a hotspot of researchers.

Surface rendering and volume rendering are the two main methods of medical image 3D reconstruction. This thesis deeply describes the theories of the two rendering methods. Then it uses CT and MRI to reconstruct the 3D image interactively.

After deeply describe the Marching Cubes surface rendering method, this thesis proposes two improvements on it. One is to reuse the shared intersected points of the cubes, it can reduce the repeated computation, save the cost of the computation, and boost the running efficiency. The other one is to use multi cores of the modern CPU, it can computes the intersected points parallelly, leading a outstanding speed of surface rendering.

On the volume rendering, this thesis first introduces the light model in computer graphics, then analyzes the method of ray casting. At last Shear-Warp acceleration is proposed.

The software platform of this thesis is based on Visual C++ and an open source image library, using the OpenGL technique to display the 3D model. This thesis proposes some improvements while describe the algorithm. Experiments are carried out under the software with contrast.

Key Words: Medical 3D Image, Surface Rendering, Volume Rendering, OpenGL, Visual C++

目录

第一章 概述

1.1 医学影像三维重建的研究背景及意义

医学影像可视化[1]是医学和计算机科学计算可视化研究领域的一个重要分支，也一直是学者研究的热门点。医学影像三维重建[2]就是把在医院里拍摄的CT（Computed Tomography－计算机X射线断层扫描技术），MRI（Nuclear Magnetic Resonance Imaging － 核磁共振成像）等数字化影像数据，通过相关算法，在计算机里直观的用三维建模的方式展现。

科学计算可视化(简称可视化，英文是 Visualization in Scientific Computing，简称 ViSC)是计算机图形学的一个重要研究方向，是图形科学的新领域[3]。

“Visualization”一词，来自英文的“Visual”，原意是视觉的、形象的，中文译成“可视化”可能更为贴切。事实上，将任何抽象的事物、过程变成图形图像的表示都可以称为可视化。但作为学科术语，“可视化”一词正式出现于1987年2月美国国家科学基金会（National Science Foundation，简称NSF）召开的一个专题研讨会上[4]。研讨会后发表的正式报告给出了科学计算可视化的定义、覆盖的领域以及近期和长期研究的方向。这标志着“科学计算可视化”作为一个学科在国际范围内已经成熟。

科学计算可视化的基本含义是运用计算机图形学或者一般图形学的原理和方法，将科学与工程计算等产生的大规模数据转换为图形、图像，以直观的形式表示出来。它涉及计算机图形学、图像处理、计算机视觉、计算机辅助设计及图形用户界面等多个研究领域，已成为当前计算机图形学研究的重要方向。

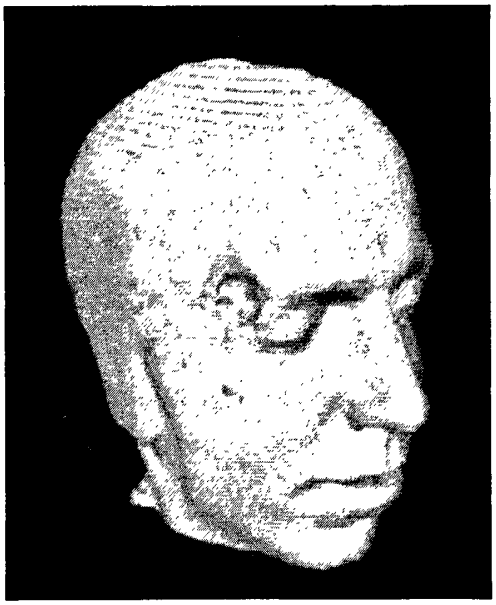


图 1.1 人体头部面绘制三维重建影像

图 1.1 即为一个典型的医学影像三维重建面绘制的例子。通过例子，可以知道，这项研究可以给医学上带来可观的价值，提高看病的效率，减少医生误判的几率。

传统的医学影像技术只是首先用 CT 等技术拍摄人体某一断层的影像数据，然后医生通过胶片或屏幕显示进行观察、诊断。但是医生观察到的只是固定视角的二维图像，这导致医生在判断病人的病情时主要依据的是所看到的图像的分析，并且需要结合医生的实际经验，有很大的主观性。

随着数字化、网络化时代的到来，医学影像技术进入一个崭新的时代。在医学影像领域，多种新的数字化成像技术(如 CT，MRI)[5]的问世，成为医学影像技术的主流。利用计算机技术对二维切片图像进行分析及处理，重建出三维模型，可以让医生多视角、多层次地进行医学病理观察和分析。从而可以大大提高医疗诊断的准确性和正确性。这给医学影像的应用增加了宝贵的价值。所以自 20 世纪 90 年代起，医学图像三维可视化技术一直是国内外研究与应用的热点。

1.1.1 医学影像三维重建在医学领域的应用

医学影像三维可视化的研究成果已经广泛的应用于医学领域。

（1）在医疗诊断和治疗中，利用医学影像可视化技术可以在计算机中定位病灶的具体位置、大小和形状，以及同周边物体的关系。这可以帮助医生更加精

确的进行医疗判断，提高诊断的准确性。

（2）医生可以利用虚拟内窥镜技术，重建出身体内部结构，在三维空间里进行内窥镜的漫游。此技术不会伤害到病人的身体，无创痛，不会造成身体病变感染。医生可以从不同角度对病灶进行观察，深入的了解病变信息。

（3）在医学教育领域，通过以三维模型重建人体的各种器官和组织，可以构建出具有真实感的三维人体，便于了解人体的身体结构。这可以增加医学教学的直观性，对教育产生巨大的作用。

1.2 医学影像三维重建的发展过程

医学影像三维可视化从上个世纪八十年代科学计算可视化作为一门独立的交叉学科到如今的迅速的发展，其发展过程可以大致分为以下三个阶段[6]：

(1) 早期探索阶段

上世纪 80 年代末可视化学科建立以前，主要针对心脏、肝脏、胚胎、神经等器官的三维重建做了很多研究；其中主要是基于表面的重建算法：轮廓提取算法、轮廓对应算法、三角面片拟合算法、曲面拟合算法等[7][8][9]；受限于当时的计算机硬件水平比如 CPU 频率比较慢，内存比较小，这一阶段的工作显得很粗糙，不能高效率的处理医学三维影像水平的图像。

(2) 基础算法研究阶段

上世纪 90 年代中期，可视化研究领域提出了基于体素的表面绘制算法，也称为间接体绘制算法：Cuberille，Marching Cubes，Dividing Cube；直接体绘制算法：RayCasting，Splatting；以及针对这两种技术的各种加速算法：其中影响最大的是由 Marc Lovoy 和 Philippe Lacrounte 于 1994 年提出的 Shear-warp 快速体绘制算法，在目前被认为是最快的三维重建直接体绘制算法；这一阶段由于计算机的进步和高效的算法的提出，重建出的医学三维影像得到了比较真实的质量和令人满意的速度。

(3) 实用研究阶段

上世纪 90 年代后期，服务于医学三维可视化技术的函数库出现，如 VTK，Volpack 等；医学三维可视化技术开始服务于外科手术模拟系统、手术导航系统、放射治疗模拟、虚拟内窥镜、外科整形、解剖模拟等临床应用。此阶段已经可以

重建出精细的三维图像，技术相对成熟。

自 90 年代起，医学影像三维重建已经在大量科研人员的潜心研究和辛苦试验下取得了巨大的成功，已经有许多的成果出现并且付诸商业应用。

在国外，已经有了三维医学影像处理的商品化系统。加拿大的独立的 Allegro 系统，它可以根据用户需要，与不同厂家的 CT 扫描设备或核磁共振仪相连接；美国通用电气公司(GE)的 AW 4.0，可以对断层扫描序列图像进行完整的容积重建及分析。有的则是这类医疗设备的一个组成部分，如以色列爱尔新特公司(ElscintUd)，GE 出产的螺旋 CT 扫描设备均附有基于图形工作站的医学影像可视化系统，在获得 CT 和 MRI 的序列扫描图像后，该系统可以沿三个正交方向逐帧显示序列图像，可以用不同方法构造三维形体，可以对三维图像由外而内按层剥离或做任意位置的剖切以观看内部结构，也可以进行平移、缩放、旋转等操作。此外，还有距离测量及面积、容积的计算等功能。很显然，具有如此强大功能的三维医学影像处理系统将给诊断和治疗提供很大的方便。但是它们需要计算速度很高、存储容量很大的计算机系统，连同软件一起，其价格非常昂贵。另外，还有美国 Stardent 计算机公司推出的 AVS 系统，美国俄亥俄超级计算机中心开发的 APE 系统，德国达姆斯达特 FHG-AGD 研究中心开发的 VIS-A-VIS 系统等。

浙江大学、清华大学、东南大学、大连理工大学、中科院自动化所等高校研究所均对医学图像三维重建及可视化研究方面做了大量研究，开发了一些实验系统，但是真正可以付诸商业应用的成熟系统目前没有[10]。

1.3 医学三维影像重建方法概述

医学三维可视化是伴随着科学计算可视化的发展进程而发展的，其经历了两个过程：基于三角面片的面绘制可视化技术和基于立方体素的体绘制可视化技术。医学三维影像可视化起始于面绘制技术，但随着计算机的不断发展，CPU 的运行效率，内存、硬盘的储存能力大幅度提高，加上有大批学者研究各种加速算法[11][12][13]，体绘制技术逐渐占领了主导地位。其绘制出来的图像的真实感，图形的细节层次都是面绘制技术无法达到的。

医学三维影像重建主要基于两种方法，一是面绘制（Surface Rendering）[14][15]，二是直接体绘制（Direct Volume Rendering）[16][17]。

1.3.1 面绘制方法概述

面绘制方法重建影像是将扫描出来的医学数据，根据一个阈值，将此阈值的等值面抽取出来，形成三维图像。并且可以加以光照模型，生成更加逼真的影像。面绘制方法的基本原理是采用曲面造型技术，通过对数据场进行解析，然后生成大量的三角面片来拟合所需要绘制的对象的表面，同时加以真实感图形学的光照明和三维消隐技术，渲染出逼真的三维图像。这种面绘制方法有一个特点就是生成的三角面片数量巨大，可以很容易达到百万级别。三角面片数量的巨大会造成计算机处理的负担很重，需要高性能的计算机来实现三维图像的旋转，平移和缩放。虽然可以运用网格逼近来缩减三角面片的数量，但是在缩减的过程中，会导致丢失医学数据细节信息，三维结果会显得比较粗糙。

常用的方法为 Marching Cubes[18][19]（移动立方体法），Marching Tetrahedra（移动四面体法）[20]等。1987 年，Lorensen 等发表了一篇论文《*Marching Cubes: A high resolution 3D surface construction algorithm*》，提出了 MC 方法，此后科学计算可视化的很多工作都以该算法为基础完成，有大量的研究工作对 MC 算法本身进行了有效的完善和提高[21]。MC 算法不仅是可视化领域的经典算法，也在图形学的其它领域中(如隐函数的显示)广为应用，因而也是图形学的基础算法之一。

1.3.2 体绘制方法概述

医学影像三维重建，要求重建出来的三维影像尽可能准确的反映对象的结构和细节。由于来自断层图像序列的数据，不同层次的图像的亮度会有细微的差别。这对于依赖判定对象间的界面进行绘制的表面重建方法非常不利。另外，由于表面绘制方法只显示等值面，对于边界明显的组织和器官反映的效果较好，而对于边界模糊的情况就不够理想了。该方法还无法反映对象内部复杂的结构。近十年来发展起来的体绘制方法，很好地解决了表面绘制方法存在的问题，以其在体数据处理及特征细节信息、表现方面的优势，已经得到研究者越来越多的重视，被广泛的应用于医学领域。

体绘制突破了面绘制方法的束缚。它不同于面绘制去寻找等值面，而是运用了计算机图形学里的光照模型，模拟光线穿过医学影像的三维数据场所产生的光

照，反射，投射等光学原理，将三维数据直接转换成二维图像而没有生成中间图形，使得整个图像的绘制一气呵成，最终形成绘制结果。直接体绘制由于涉及到光照模型等复杂计算，所需要的计算时间也长，基本不能实时处理。1994 年，Philippe Lacroute 和 Marc Levoy 共同发表了《*Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*》论文，文中将 Shear-Warp 方法用于体绘制方法，运用错切变换，加速了体绘制的过程[22]。

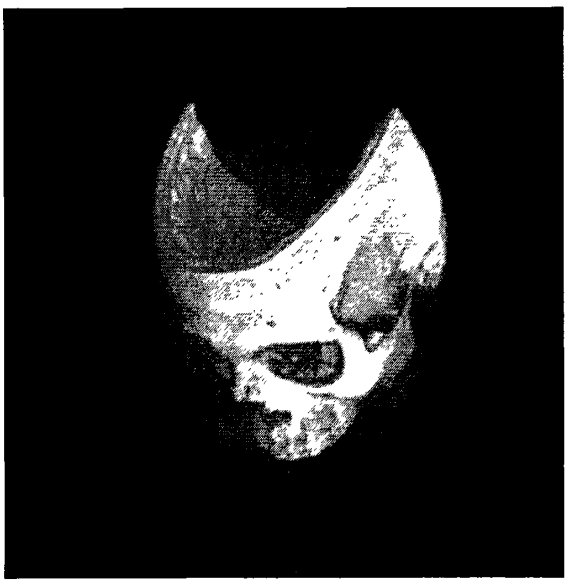


图 1.2 通过体绘制方法绘制出来的人体头颅影像

随着科技的进步和计算机处理能力的提高，想要得到精度非常大的医学数据都可以，所以医学三维影像的最终显示效果也越来越好。不仅可以重建出外表影像，内部三维模型也可以重建出来，比如人体血管内部跟踪，都是可以实现的。

1.4 本文的研究内容

本文主要深入研究面绘制和体绘制的原理和方法，然后由医学图像（CT、MRI）的二维断层序列来重建三维图像并与之交互。应用 Visual C++和开源图形图像处理包对图像断层序列进行计算机处理，用 OpenGL 技术将结果用三维模型显示，完成重建。同时通过对面绘制的算法进行一些改进，提高面绘制的效率，并且在 OpenGL 画图时运用其 GPU 加速的特性，加速了整个绘制过程。本文在阐述算法的同时，提出了相应的改进，最后通过在自己创建的软件平台下，进行试验，得

到了对比的结果。

本文的章节安排如下:

第一章: 概述医学影像三维重建，从其研究背景，研究目的，临床应用和发展过程出发，对医学影像三维重建进行概要描述。

第二章: 详细研究了面绘制方法。深入介绍面绘制方法中 Marching Cubes 方法的原理和二义性。同时提出了基于立方体棱边共享等值点的加速改进和基于多线程并行计算的加速改进，并给出了试验对比结果和分析。

第三章: 详细研究了体绘制方法。在介绍了光照明模型和光线投射算法之后，阐述了经典的 Shear-Warp 体绘制方法。并给出了实际的例子。

第四章: 深入介绍了本文的算法软件平台。该平台是基于 Visual C++ 中的 MFC 结合 OpenGL 技术来实现医学影像三维重建算法。操作简单，生成的三维图像直观，易懂。

第五章: 总结医学三维影像可视化方法，提出未来的展望。

第二章 医学影像三维重建面绘制改进和加速方法

2.1 面绘制概述

对于 CT 或者 MRI 扫描出来的人体器官或组织的影像数据，通过分析可以发现，不同的器官有着不同的灰度值。特别地，有些器官外形分明，比如心脏，如果能把跟心脏灰度值一样的面片都找到，那么就可以在计算机中把 CT 或 MRI 扫描出来的二维心脏数据，经过算法处理，重建为三维心脏数据，各个角度都可以旋转，查看。面绘制的本质就是抽取空间体素里的对应特定阈值的等值面。由于面绘制的算法较为简单，因此在现代计算机中，可以做到实时绘制。不过，如果有些器官轮廓不清晰明朗，或者器官不仅仅是一个面，有多个面包含等复杂的方式，面绘制就不能达到很好的效果[23]。

1987 年，Lorensen 等发表了一篇论文《*Marching Cubes: A high resolution 3D surface construction algorithm*》，提出了著名的 MC 方法，翻开了面绘制技术的新封面。它通过对空间三维体素网格的遍历，抽取等值面，进行重建出三维医学图像。

2.2 医学数据集

现在去医院拍片已经是极为常见的事情了。在医学上，可以通过 CT，MRI 等技术获取二维断层图像。通常可以得到一系列的断层图像，如人体切片，就是一系列平行的二维图像。其本质是对人体进行离散采样进而绘制出一张图片，正是因为这个原因，使得三维重建可以进行。国外已经制定出了完善的医学图像格式标准，如著名的 DICOM。医学数字图像通讯标准 DICOM3. 0 (Digital I maging and Communications in Mdicine) 是最近几年在医学信息领域蓬勃兴起的一种通用的与生产厂商无关的图像。DICOM 作为医学影像通讯的统一接口，给医学影像的发展起到了很大的作用[24]。

对于一个断层图像，由于其是二维的图像，所以在计算机中本质是一个矩阵，由于离散的特性。二维图像的最基本单位是像素，比如一个 CT 拍出来的图像，其分辨率是 512*512，说明水平和竖直方向都有 512 个像素点，这些像素点就是

我们可以用来重建三维图像的最基本的元素。在MC算法中，前后左右上下相邻的8个像素就可以作为一个小立方体。

断层图像除了有像素分辨率以外，还有一个重要特征就是每一个像素所拥有的值，通常叫做灰度值。如果图像是8位（bit），那么灰度值就有2^8=256阶。不同的人体组织往往拥有不同的灰度值，所以通过重建指定灰度值的等值面，就实现了三维重建。

综上，通过CT，MRI扫描得到的二维图像最小单位是像素，一系列图像构成的数据集的最小单位叫做体素（即8个相邻的像素组成的立方体）。

2.3 等值面

等值面是指体数据中具有相同性质的点的集合。由于不同的物质具有不同的物理属性所以可以选取适当的值确定等值面，用等值面表示不同物质的交界面。体素级重建方法不会遇到基于轮廓重建的轮廓对应、轮廓拼接和分叉问题。因此在医学图像三维重建得到了广泛地应用[25]。

所有满足$f(x,y,z)=C$的像素点组成了等值面。

2.4 Marching Cubes方法

Marching Cubes（移动立方体）方法本质上就是从三维的数据场（或称体素）中抽取等值面（由多边形面片组成）的算法。在二维的时候，叫做Marching Square（移动正方形）方法。

该算法逐个遍历数据场，每8个顶点组成一个立方体为最小处理单元，然后确定该立方体上是否有等值面，如果有，计算等值面与立方体的棱边的顶点。这样按照一定顺序把所有立方体遍历，最终可以得到所想要的曲面（等值面）。

一个标准的医学影像数据集是由一系列的断层图像所组成的。假设现在有一组数据，包含100张图片，每张图片的分辨率为256*256，对应的采集的数据集恰恰是一个连续函数$f(x,y,z)$在空间三个方向按一定的间隔所进行的离散采样。当然图片上的采样点的灰度值是各不相同的，令$f(x,y,z)=C$，C是一个固定值，称其为阈值，这样在几何上，这个方程就表示了一个曲面。等值面就是这样一种

阈值都是固定值的点的集合，MC 算法就是要从数据集中找出所设置的阈值的等值面。

由于等值面是三次代数曲面，如果从正方向比如插值去求等值面，势必会导致很复杂的计算。MC 方法是从反方向，不计算 $f(x,y,z)$，而是确定出等值面与体素的交点，把交点连成三角面片，最后组合为等值面。

MC 方法有个基本前提：沿着体素的棱边数据场呈线性变化。也就是指，在一个立方体体素里，一条边的两个顶点的数据场的值一个大于、另一个小于等值面的值，那么该边上有且只有一个与等值面相交的点，称作等值点。MC 方法就是先计算出每一个体素的等值点，按规定的方式把等值点连接成三角面片作为等值面的逼近表示，同时可以运用差分法计算三角面片的法向量，作为模拟光照的必备要素。

首先讨论怎么样求等值点。

由于一个体素有 8 个顶点，每个顶点有相应的数据场值。我们对 8 个顶点进行处理，得到一个体素的特征，进行相应归类。

假定我们所要求的等值面的阈值为 C，如果该顶点的值大于等于 C，将该顶点“涂黑”，如果该顶点的值小于 C，不做处理。

记一条边上的顶点坐标分别为 $P1$，$P2$，他们对应的数据场值分别为 $V1$，$V2$，等值点的坐标为 P，那么由下面的公式可以求出等值点的坐标：

$$P = P1 + (P1 - P2) * \frac{C - V1}{V1 - V2}$$

同理，假设法向量用 N 表示，那么下面的公式求出等值点的法向量：

$$N = N1 + (N1 - N2) * \frac{C - N1}{N1 - N2}$$

由于体素网格分辨率很高，所以如果直接简化求等值点，把等值点看做棱边的中点，也是可行的，在计算机显示出来没有明显的差异。具体如下：

$$P = \frac{1}{2} * (P1 + P2)$$

$$N = \frac{1}{2} * (N1 + N2)$$

现在考虑等值面与体素相交的情况，由于每个顶点有 2 个可能性，要么在等值面内，即数据场值小于阈值，要么在等值面外，即数据场值大于阈值。一个体素的 8 个顶点一共可能有 2 的 8 次方即 256 种情况，这是一个比较大的数字[26]。

不过，在87年那篇论文中，作者提到立方体有:

1. 旋转对称性，立方体经过旋转不影响等值面的拓扑结果

2. 对称不变性，将涂黑的点取消，将未涂黑的点涂黑，拓扑结构依然不变。

所以，作者总结出一共有以下15种拓扑结构:

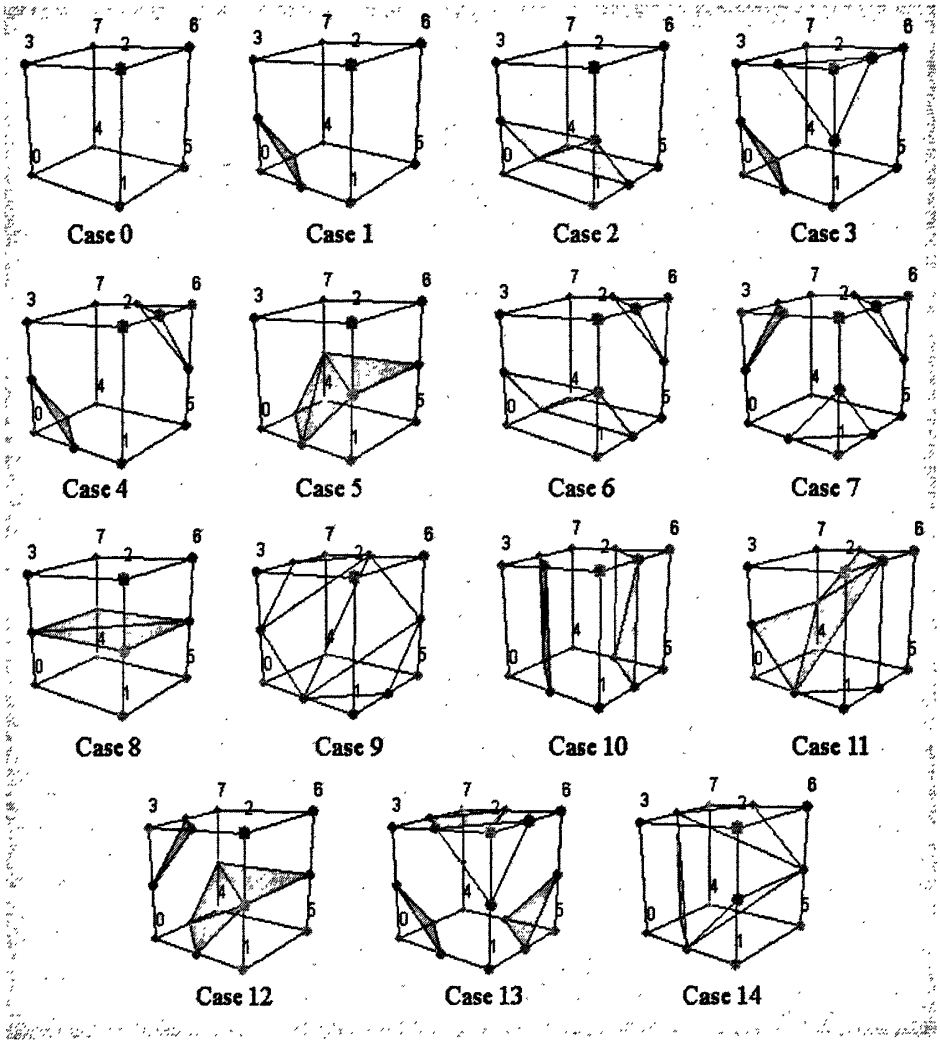


图2.1 15种拓扑结构，红色顶点表示在等值面内，蓝色顶点表示在等值面外

这样，通过对体素的数据场值判断，就可以分析出该体素属于哪种情况。算法如下:

用一个8位（1字节）的变量来保存一个体素的状态，第0位表示第0个顶点状态，如果小于阈值则赋值为0，大于阈值则赋值为1。其余顶点也是如此。这个8位的变量最后的值刚好是0到255，对应着一个体素分类。在计算机程序中以下面程序片段可以实现:

```
unsigned char cubeindex = 0;
if (grid.val[0] > isolevel) cubeindex |= 1;
if (grid.val[1] > isolevel) cubeindex |= 2;
if (grid.val[2] > isolevel) cubeindex |= 4;
if (grid.val[3] > isolevel) cubeindex |= 8;
if (grid.val[4] > isolevel) cubeindex |= 16;
if (grid.val[5] > isolevel) cubeindex |= 32;
if (grid.val[6] > isolevel) cubeindex |= 64;
if (grid.val[7] > isolevel) cubeindex |= 128;
```

经过以上程序的处理，一个 grid 的 8 个顶点都赋予了 0 或者 1，8 个顶点共同作用的值，反映到 cubeindex 上用一个 0 到 255 的整数来表示。

2.5 Marching Cubes 方法的二义性

MC 方法归类出来的 15 种情况，并不是完美的，这里，我们先回到二维的情况（当然，只看体素的一面也可以）。

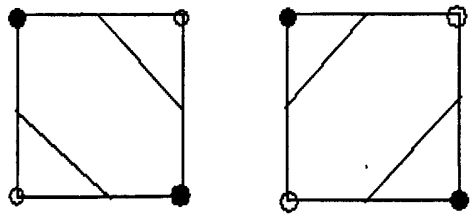


图 2.2 图示为存在二义性的 2 个情况

从图上可以知道，当一个体素的面的一对对角顶点被涂黑，而另一对是未涂黑的，就会出现二义性。此时如果没有进一步的条件，上图的左、右两个选择都是可以的。

如果两个相邻的体素的恰好都是上面的类型，那么最后重建出来的图像可能会有空洞。拿二维举个例子：

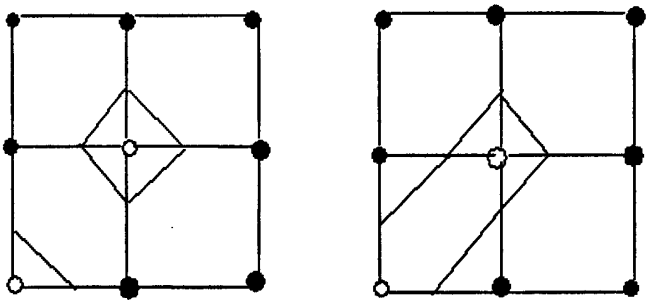


图 2.3 二义性曲线不连续

上图的左边和右边的对比，可以看出，顶点状态都是一样的，但是连接方式不一样，导致曲线不连续，放到三维上来说，就是会让等值面出现空洞。

2.6 二义性的解决方法

那么怎么样解决这个问题呢？一个最直观的想法是继续细分此网格，直到每个小体素不会再出现二义性．相应的，增加 CT 拍摄的分辨率，也可以同样相对的达到使网格更细密的效果，从而消除二义性。

1991 年，Nielsen 等人发表论文《The asymptotic decider: resolving the ambiguity in marching cubes》，提出了 asymptotic decider 方法[27]，它是最为常用最方便的一个判断面二义性属于哪种情况的方法，其基本原理如下：通过计算等值面与立方体素边界的交线（双曲线）的渐近线与体素的边界的相互位置关系来判断等值面的正确连接方式．当出现面二义性的时候，双曲线的渐近线的交点应该和其中没有在等值面内的一对顶点的状态是一样的。

可以用下图来表示双曲判定的方法：

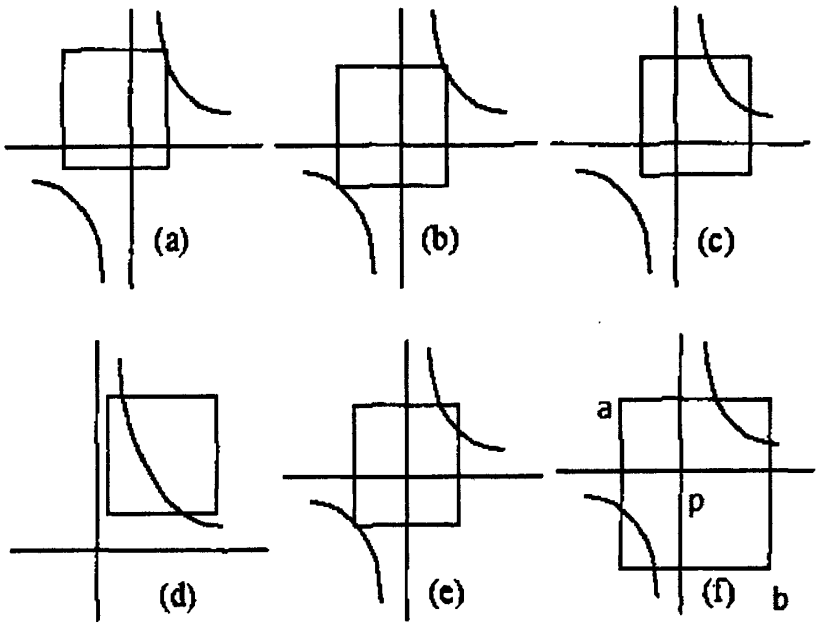


图2.4

图 2.4 里的 f 图就是一个典型的出现面二义性的例子. 此时应该满足的条件为

$$f(a), f(b), f(p) > C$$

即此三个点都处在等值面的同一侧.

用 asymptotic decider 后，并不会产生更多的三角面片，配合 OpenGL 里的多边形三角化 API，可以实现正确的构造等值面.

2.7 移动四面体法

由于移动立方体有二义性，所以发展出了另一种方法，叫做移动四面体(Marching Tetrahedrons).

如下图所示，等值面穿过四面体只有如下三种拓扑结构，不会出现二义性.

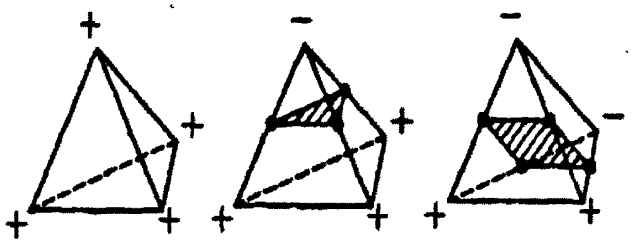


图2.5

因此，把立方体分为四面体之后，演变出来的 MT 方法，可以避免二义性，使得重构出来的图像更加完美．当然，由于细分，相应的由四面体割出来的三角面片也增加[28]．

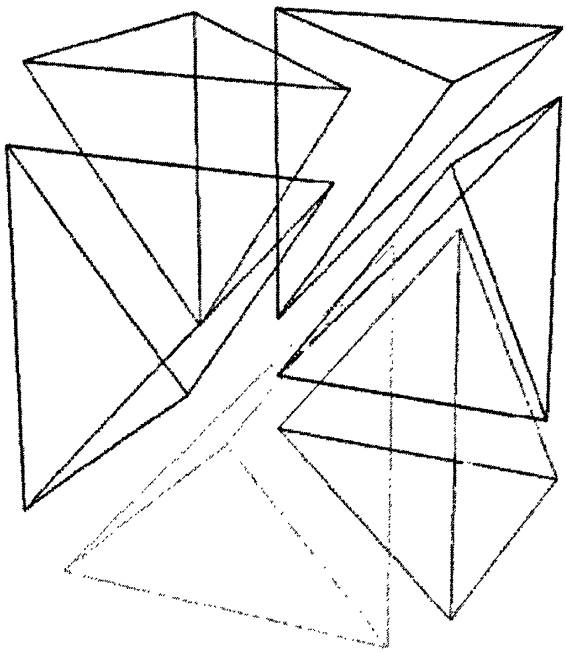


图2.6 立方体分解为6个小四面体

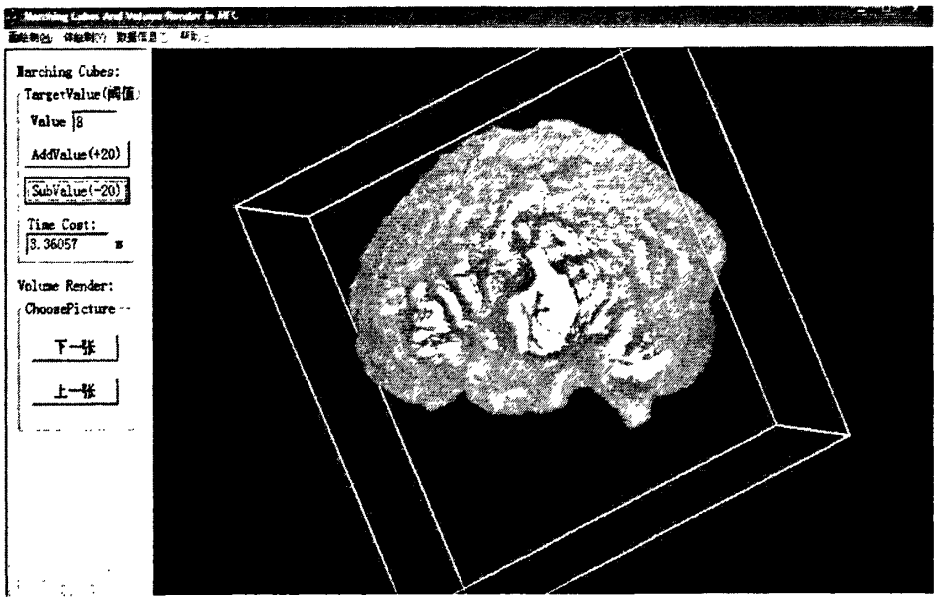
2.8 MC 算法实现示例

医学三维影像重建的目的就是在计算机中展现出结果，所以本文的研究内容之一就是写出程序来实现算法，将结果在计算机中展示。

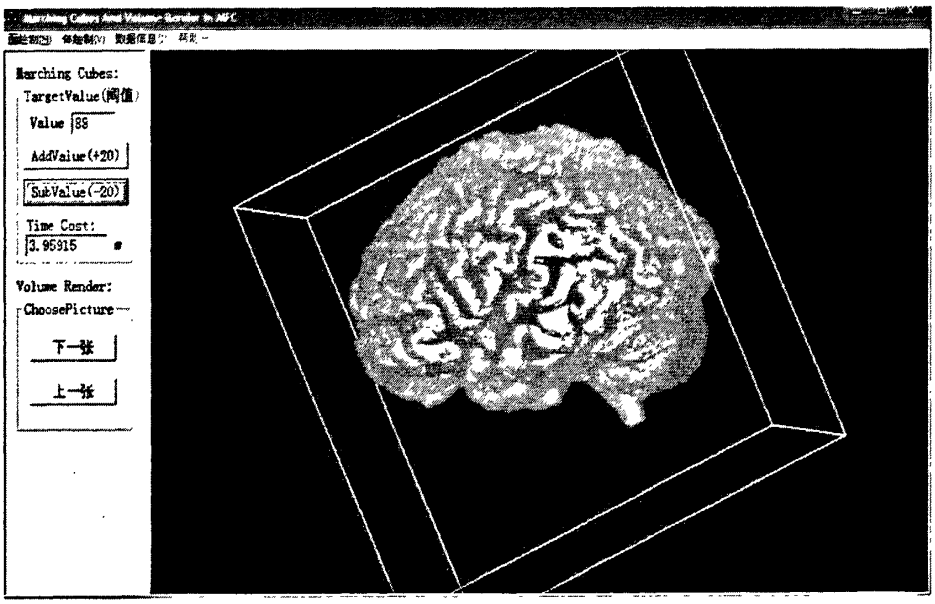
本文选择 Windows 平台的 Visual C++集成开发环境配合 OpenGL 这一强有力的三维图形开发包来进行医学三维影像可视化的开发。

首先以人的大脑三维数据进行重建，提供几种不同阈值的重建结果进行对

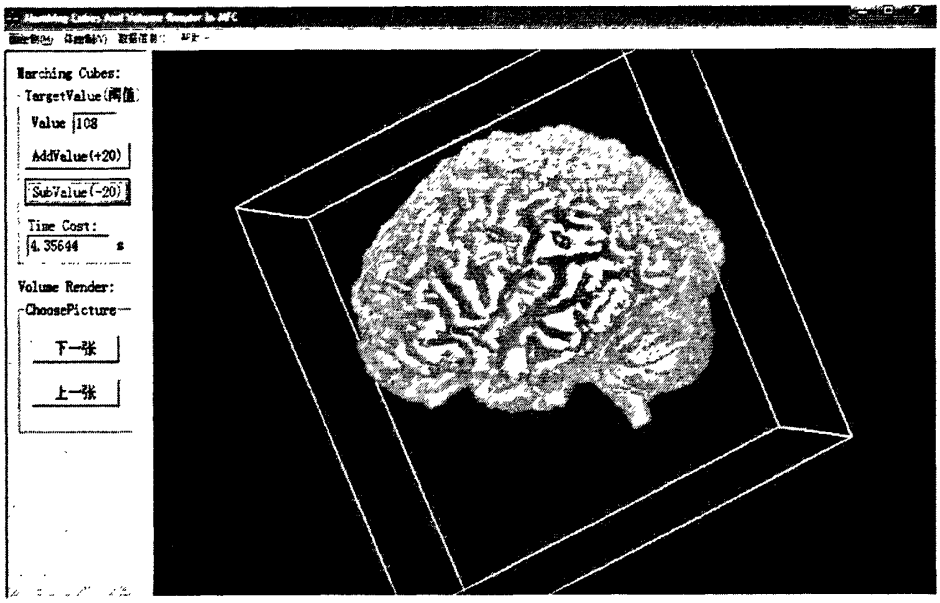
比。



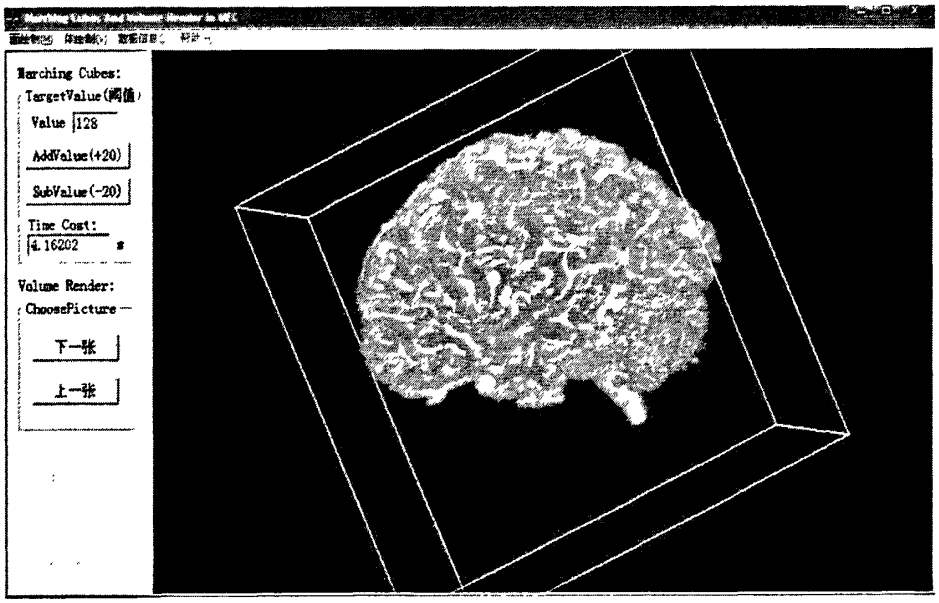
(1) 阈值为 8 的重建图



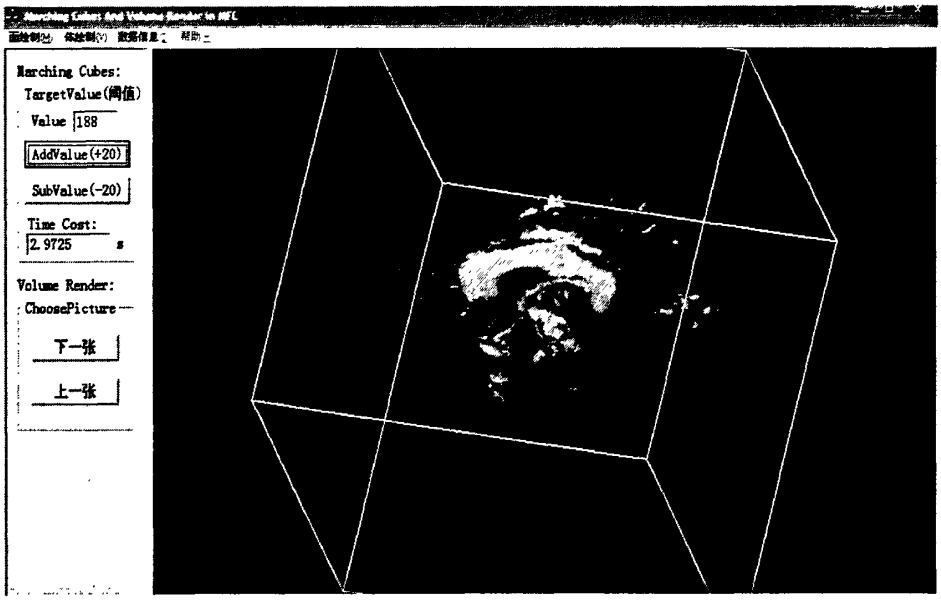
(2) 阈值为 88 的重建图



(3) 阈值为 108 的重建图



(4) 阈值为 128 的重建图



(5) 阈值为 188 的重建图

通过输入原始数据 brain.raw，通过算法处理，生成三角面片的顶点信息和光照信息，配合 OpenGL 三维建模，最终得到上面的几幅图。外面的立方体框表示图形所在的三维立方空间。可以通过旋转来观察不同角度的图像。

分别截取了阈值为 8，88，108，128，188 时重建出的三维图。可以看到，阈值不同，所生成的结果也不同，从大脑的细节就可以看出来。

当然，由于不同阈值，所以算法处理的立方体体素，生成的三角面片也不一样多，消耗的时间不同。在阈值为 188 的时候，由于生成的三角面片数量骤减，故时间仅需要 2.9725 秒。同 108 阈值的 4.35644 秒有很大差别。

2.9 对 MC 算法的改进

2.9.1 基于立方体棱边共享等值点的加速改进

虽然 MC 算法很直观，即对每个立方体进行处理，通过 8 个顶点计算该立方体的索引值，然后根据索引值去查出哪些棱边有交点，通过线性插值运算并求出棱边上的交点，绘制出三角面片。

但是，对一个立方体的棱边来说，它不仅仅属于当前的这个立方体，它同时还属于另外三个与当前立方体相邻的立方体。即一条棱边是有 4 个立方体共享的。若按照朴素的 Marching Cubes 算法进行计算，那么如果 4 个立方体都是在

这棱边有等值点的话，计算会重复 4 次，大大浪费了计算机的处理开销，增加了算法的时间。

基于以上原因，本文提出如下改进来提高 Marching Cubes 算法的效率，实践证明此改进可以提高算法效率 20%到 30%。

我们在找到有等值点的棱边之后，不是立即通过线性插值来计算等值点的坐标，而是用一个全局数组变量保存一个立方体里的每条棱边的相关信息，包括这条棱边是否有等值点，如果有等值点，等值点的值是多少，同时也记录了等值点的法向量。

在计算等值点之前，先判断此棱边是否有等值点，如果有，则直接把已经存在数组里的数据提取出来用于接下来的流程。如果查询显示此棱边没有等值点，才进行插值计算等值点的坐标和法向量。算法最为关键的是，在这一步，不仅仅只是把当前体素 grid[i][j][k]的棱边 edge 的等值点给记录下来，还要把共享这条边的其他的体素的棱边 edge 的等值点给予赋值，这样才可以在接下来的算法中查询到此棱边已经有等值点，不用重复计算。

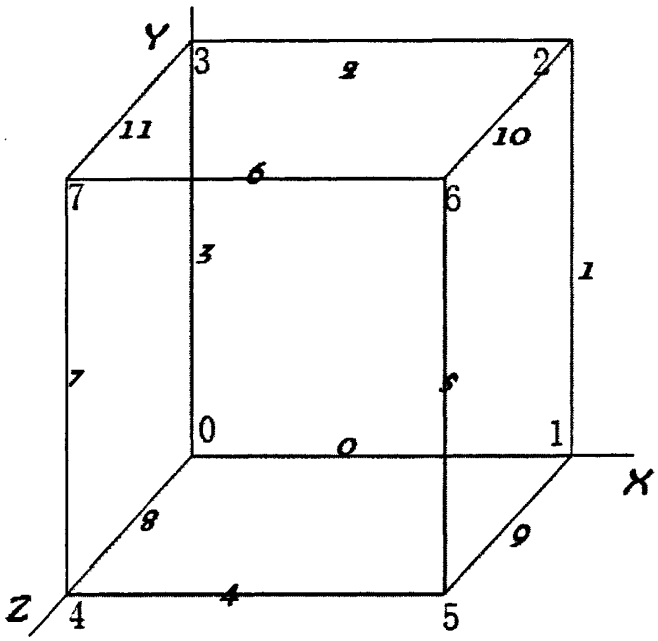


图 2.7 一个立方体的顶点和棱边的编号

假设图 2.7 的立方体是在 X 方向的第 i 个，Y 方向的第 j 个，Z 方向的第 k 个，即 grid[i][j][k]。由于我们的算法是按照 x，y，z 方向增加的，那么必然

只需要考虑三个面的 9 条棱边即可，[1 10 5 9]，[2 10 6 11]，[6 5 4 7]这三个面是会与还没处理的立方体有共享棱边等值点，一共有编号为 1 10 5 9 2 4 11 6 7 的 9 条棱边。下图展示的是 x 方向伸展的由棱边[1 10 5 9]组成的共享面。左边的立方体标出了顶点和棱边的编号，相应的，右边的立方体的顶点和编号虽然和左边的不同，但是有对应的关系。例如左边的立方体的棱边 1 是右边的立方体的棱边 3。

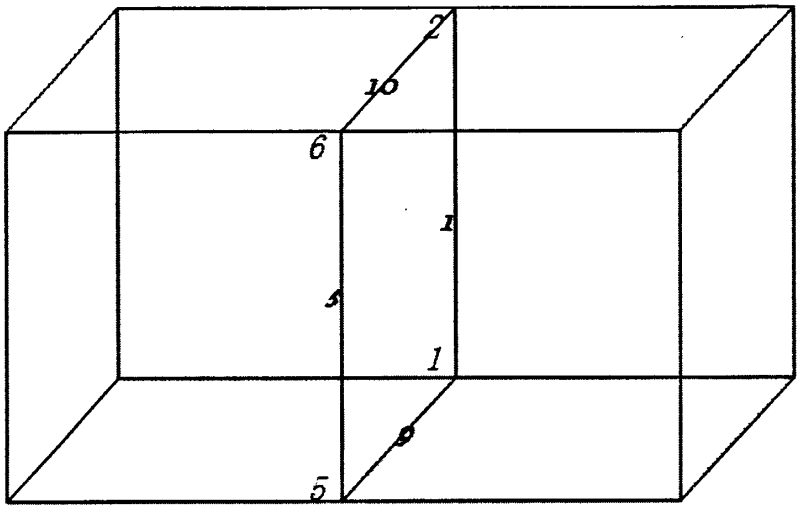


图 2.8 两个立方体体素的共享面和共享边

下面是 9 条边所对应的其他立方体棱边等值点的信息:

棱边 1: grid[i+1][j][k]的棱边 3 有相同等值点

棱边 10: grid[i+1][j][k]的棱边 11 有相同等值点，grid[i][j+1][k]的棱边 9 有相同等值点

棱边 5: grid[i+1][j][k]的棱边 7 有相同等值点，grid[i][j][k+1]的棱边 1 有相同等值点

棱边 9: grid[i+1][j][k]的棱边 8 有相同等值点

棱边 2: grid[i][j+1][k]的棱边 0 有相同等值点

棱边 4: grid[i][j][k+1]的棱边 0 有相同等值点

棱边 11: grid[i][j][k+1]的棱边 8 有相同等值点

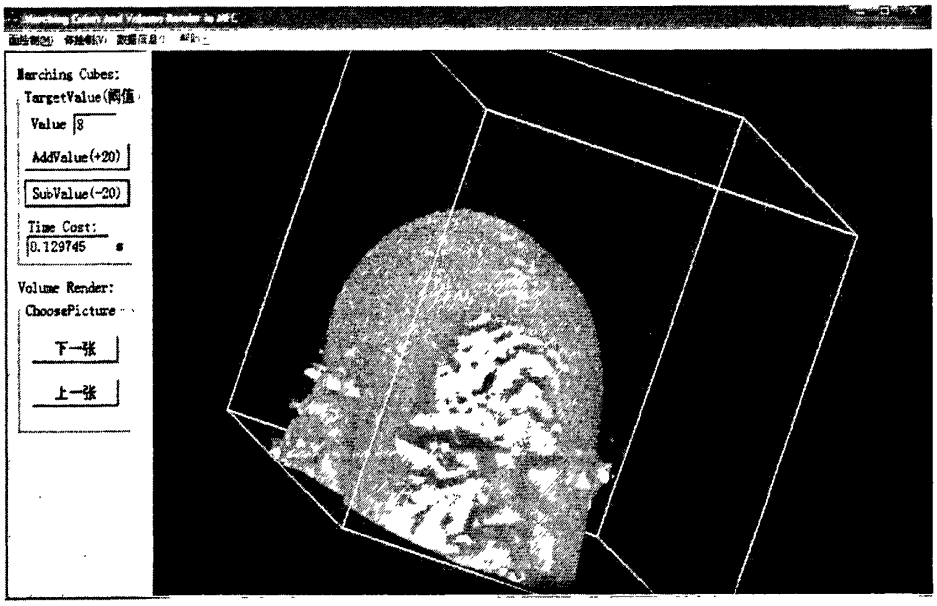
棱边 6: grid[i][j+1][k]的棱边 4 有相同等值点，grid[i][j][k+1]的棱边 2 有相同等值点

棱边 7: grid[i][j][k+1]的棱边 3 有相同等值点

通过对每一次计算等值点时，加上以上的算法，可以使得不用重复插值计算棱边的等值点，4 次计算简化为 1 次计算，大大提高了算法的效率。

本文把以上算法运用到程序代码之中，通过实际的面绘制实例，验证了提出的改进算法的效果。通过将一个维度为 48*62*42 的医学扫描数据进行面绘制重建（阈值为 8），得到了以下对比信息:

	生成面片数	计算交点个数	共享交点个数	算法时间(秒)
未改进	33284	68128	0	0.129745
改进	33284	34095	34033	0.10652



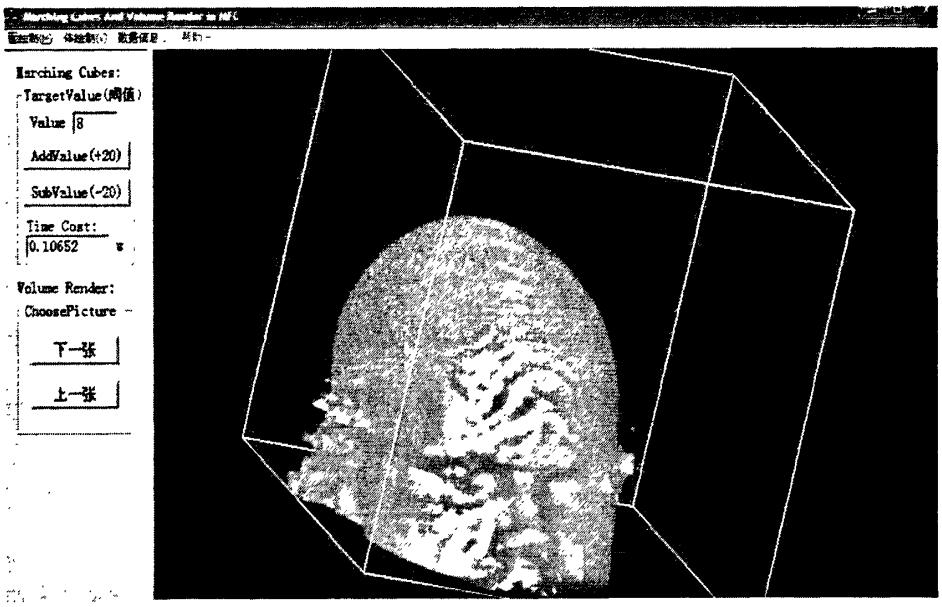


图 2.9 加速前与加速后的程序运行面绘制的时间截图对比

2.9.2 基于多线程并行计算的加速改进

每个正在系统上运行的程序都是一个进程。每个进程包含一到多个线程。进程也可能是整个程序或者是部分程序的动态执行。线程(Thread)是一组指令的集合，或者是程序的特殊段，它可以在程序里独立执行。也可以把它理解为代码运行的上下文。所以线程基本上是轻量级的进程，它负责在单个程序里执行多任务。通常由操作系统负责多个线程的调度和执行。

线程是程序中一个单一的顺序控制流程.在单个程序中同时运行多个线程完成不同的工作,称为多线程.

多线程主要是为了节约CPU时间,发挥利用,根据具体情况而定. 线程的运行中需要使用计算机的内存资源和CPU。

多线程是为了同步完成多项任务，不是为了提高运行效率，而是为了提高资源使用效率来提高系统的效率。线程是在同一时间需要完成多项任务的时候实现的。

对于 Marching Cubes 算法，由于算法的最小处理单位为一个体素，仅仅计算一个体素内的等值三角面片，计算过程不用依赖其他立方体(当然，前面提到可以利用共享棱边等值点)。所以，Marching Cubes 天然就是一个可以用多线程进行并行计算的算法。

下面是流程图对比:

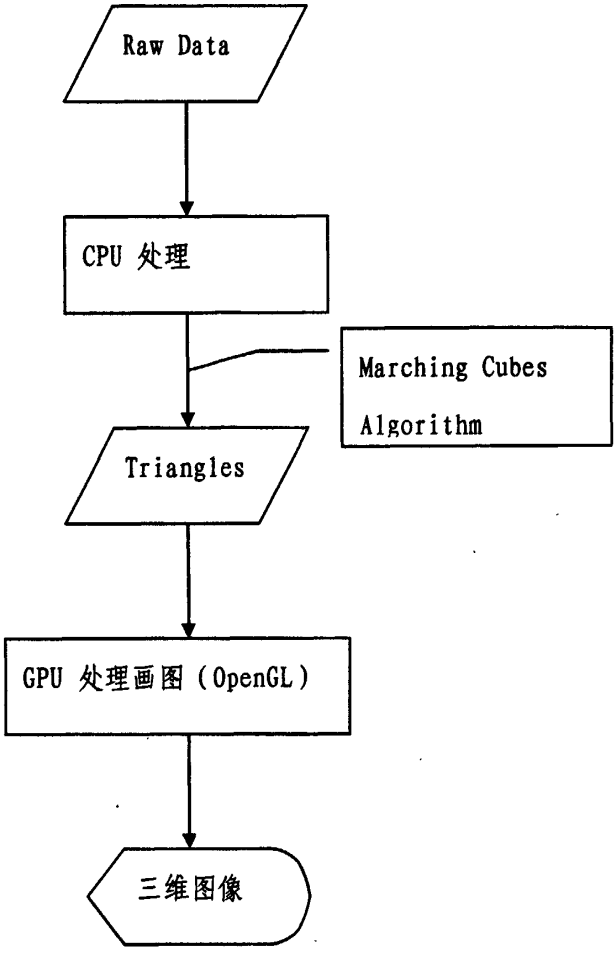


图 2.10 单线程 Marching Cubes 算法流程示意图

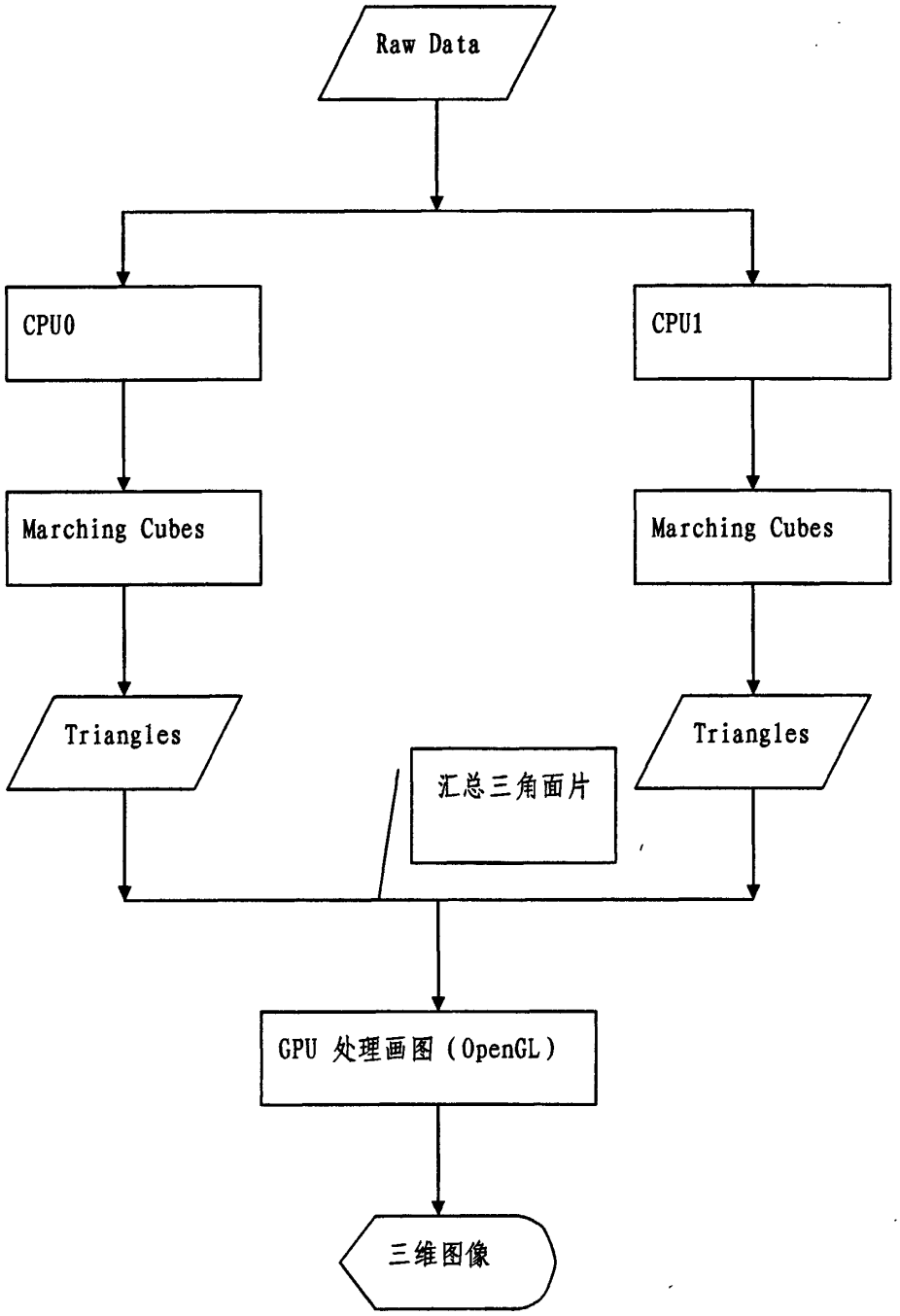


图 2.11 双线程 Marching Cubes 算法流程示意图

由于双核 CPU 在现在已经是主流，早已普及，所以本文的程序编写也是在一台 AMD 双核 CPU，型号为 AMD Athlon 64 X2 4000+的计算机上实现的。由于整个

算法框架是在MFC（Microsoft Foundation Classes）上编写的，所以软件可以实现双线程Marching Cubes的同时运算，然后将计算出来的三角面片汇总，通过GPU加速从而绘制出三维图形。

线程的触发需要调用AfxBeginThread函数，函数原型如下：

CWinThread* AfxBeginThread(AFX_THREADPROC *pfnThreadProc*, LPVOID *pParam*, int *nPriority* = THREAD_PRIORITY_NORMAL, UINT *nStackSize* = 0, DWORD *dwCreateFlags* = 0, LPSECURITY_ATTRIBUTES *lpSecurityAttrs* = NULL);

通过指定2个线程函数thread1，thread2来启动双线程：

AfxBeginThread(thread1, this, THREAD_PRIORITY_ABOVE_NORMAL);

AfxBeginThread(thread2, this, THREAD_PRIORITY_ABOVE_NORMAL);

同2.8.1进行的绘制环境一样，运行了双线程并行计算之后的算法，时间又提高了20%左右。只需要0.818861s即可绘制完毕。

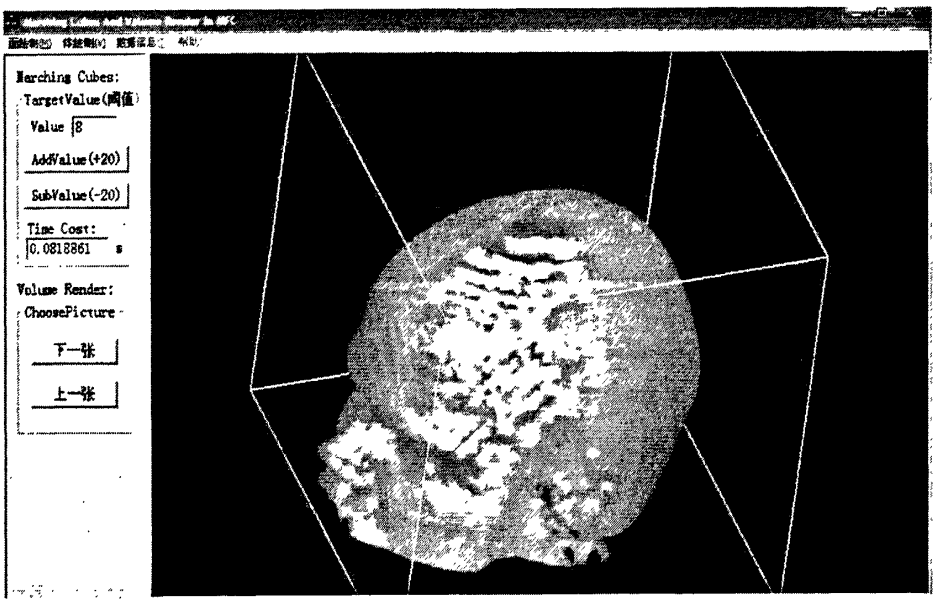


图2.12 多线程并行计算之后的绘制时间 0.0818861s

同2.8.1里所提到的算法的对比：

	生成面片数	计算交点个数	共享交点个数	算法时间(秒)
未改进	33284	68128	0	0.129745
2.8.1改进	33284	34095	34033	0.10652
2.8.2改进	33284	34095	34033	0.0818861

可以看到2.8.2在2.8.1的基础上又加快了20%左右。充分显示了并行计算的效果。由于程序每一次运行的时间都会有稍微的不同，故文中所提到的数据仅为当时运行的结果。

第三章 医学影像三维重建体绘制方法

上一章讲的是三维重建的面绘制方法，虽然算法简单，效率很高，但是生成的三维图像是通过几何三角面片组成的。同时，由于面绘制的本质，决定了只能生成物体的轮廓，要想把物体内部的结构也展现出来，就很难用面绘制方法来精确呈现。比如某些软组织，其内部结构是复杂而细腻的，要如何精确的重建出这类医学组织呢？如果用面绘制，就会只能展现某一阈值的面，舍弃很多内部信息。

于是研究人员结合计算机图形学，发展出了直接体绘制（Direct Volume Rendering）方法，它把三维体数据，直接通过图形学计算，投影到二维图像。它不需要向面绘制方法那样生成中间图元，而是直接计算图像每个像素点的颜色值，进而生成整个高质量，真实感投影图像[29][30]。

由于此章节需要真实感图形学的基础，故先介绍真实感图形学的一些基础知识[31]。

3.1 简单光照明模型

当光照射到物体表面时，光线可能被吸收、发射和透射。被物体吸收的部分转化为热，反射、透射的光进入人的视觉系统，使我们能看见物体。为模拟这一现象，我们建立一些数学模型来替代复杂的物理模型，这些模型就称为明暗效应模型或者光照明模型。三维物体的图形经过消隐后，再进行明暗效应的处理，可以进一步提高图形的真实感。

光照明模型是很多科学研究者通过不断努力改进得来的。

1967 年，Wylie 等人第一次在显示物体时加进光照效果。Wylie 认为：物体表面上一点的光强，与该点到光源的距离成反比。

1970 年，Bouknight 在 Comm.ACM 上发表论文，提出第一个光反射模型，指出物体表面朝向是确定物体表面上一点光强的主要因素，用 Lambert 漫反射定律计算物体表面上各多边形的光强，对光照射不到的地方，用环境光代替。

1971 年，Gourand 在 IEEE Trans.Computers 上发表论文，提出漫反射模型加插值的思想。对多面体模型，用漫反射模型计算多边形顶点的光亮度，再用增量法插值计算。

1975 年，Phong 在 Comm. ACM 上发表论文，提出图形学第一个有影响的光照明模型。Phong 照明模型虽然只是一个经验模型，但是其真实度已经达到可以接受的程度。

1977 年，Blinn 从物理学角度提出了一个更为准确的光照明模型。实际中发现这种光照模型与 Phong 光照模型生成的图像没有多大的差别，而计算上又比较费时，所以实际运用中一般仍然采用 Phong 光照明模型。

1980 年，Whitted 提出了整体光照明模型，他在 Phong 模型中增加了环境镜面反射光亮度和环境规则透射光亮度。

1983 年，Hall 引入了规则透射高光以及模拟整体镜面反射和透射光在传输介质中的衰减函数，进一步改进了 Whitted 模型。

1984 年，Goral 等人提出使用热辐射工程中的辐射度方法，解决了光能在景物表面间由漫反射到漫反射的能量传播形式。

真实感图形学的发展，使得医学影像三维重建可以利用此项技术，重建出更加逼真的图像。本文论述光线投射法和 Shear-Warp 法。

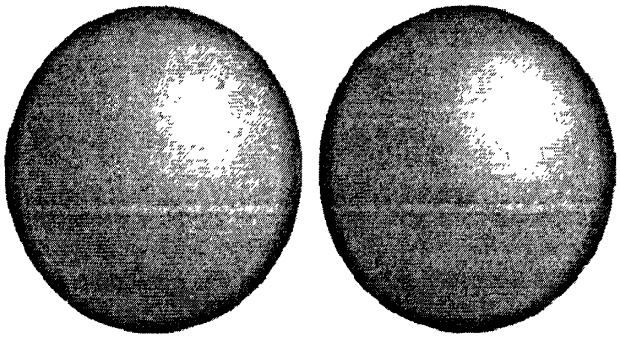


图 3.1 运用 Phong 模型在计算机中生成的小球光照模型

3.1.1Phong 光照明模型

光照到物体表面时， 物体对光会发生反射(Reflection)、透射(Transm--ission)、吸收(Absorption)、衍射(Diffraction)、折射(Refraction)、和 干涉(Interference)。首先我们先来介绍对于光反射现象的研究。

简单光照明模型模拟物体表面对光的反射作用。光源被假定为点光源， 反射作用被细分为镜面反射(Specular Reflection)和漫反射(Diffuse Reflection)。简单光照明模型只考虑物体对直接光照的反射作用，而物体间的

光反射作用，只用环境光(Ambient Light)来表示。Phong 光照明模型这样的一种模型。下面，我们分别从光反射作用的各个组成部分来介绍这个简单光照明模型。

1. 理想漫反射

当光源来自一个方向时，漫反射光均匀向各方向传播，与视点无关，它是由表面的粗糙不平引起的，因而漫反射光的空间分布是均匀的。记入射光强为 I_p，物体表面上点 P 的法向为 N，从点 P 指向光源的向量为 L，两者间的夹角为 θ，由 Lambert 余弦定律，则漫反射光强为:

$$I_d = I_p K_d \cos\theta, \theta \in (0, \pi/2)$$

其中，K_d 是与物体有关的漫反射系数，$0 < K_d < 1$。

2. 镜面反射光

对于理想镜面，反射光集中在一个方向，并遵守反射定律。对一般的光滑表面，反射光集中在一个范围内，且由反射定律决定的反射方向光强最大。因此，对于同一点来说，从不同位置所观察到的镜面反射光强是不同的。镜面反射光强可表示为:

$$I_s = I_p K_s \cos^n(\alpha), \quad \alpha \in (0, \pi/2)$$

其中 K_s 是与物体有关的镜面反射系数，α 为视线方向 V 与反射方向 R 的夹角，n 为磨光指数，反映了物体表面的光泽程度，一般为 1～2000，数目越大物体表面越光滑。镜面反射光将会在反射方向附近形成很亮的光斑，称为高光现象。

3. 环境光

环境光是指光源间接对物体的影响，是在物体和环境之间多次反射，最终达到平衡时的一种光。我们近似地认为同一环境下的环境光，其光强分布是均匀的，它在任何一个方向上的分布都相同。例如，透过厚厚云层的阳光就可以称为环境光。在简单光照明模型中，我们用一个常数来模拟环境光，用式子表示为:

$$I_e = I_a K_a$$

其中：I_a 为环境光的光强，K_a 为物体对环境光的反射系数。

4. Phong 光照明模型

综合上面介绍的光反射作用的各个部分，Phong 光照明模型有这样的一综合上面介绍的光反射作用的各个部分，Phong 光照明模型有这样的一想漫反射光强 I_d、和镜面反射光 I_s 的总和，即：

$$I = I_a K_a + I_p K_d (L \bullet N) + I_p K_s (R \bullet V)^n$$

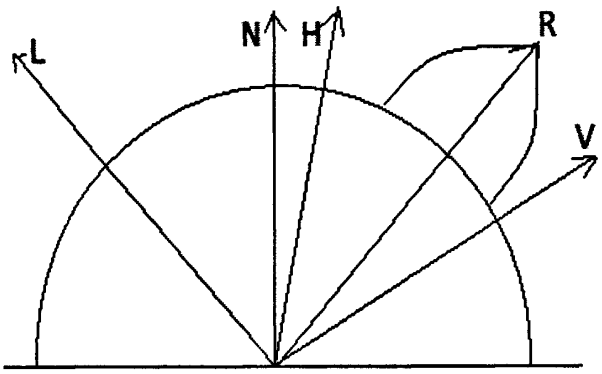


图 3.2 Phong 模型中的几何量示意图

3.2 光线投射体绘制算法

3.2.1 光线投射背景

光线投射（Ray Casting）算法为图形学中的经典算法[32]。它模拟一个点光源，发出光线，达到物体所产生的镜面反射光和漫反射光，并且可以加入透射（用阻光度控制），最终形成物体上点的颜色。它不同于光线跟踪，或者说它是光线跟踪算法的简化。因为光线跟踪算法，要考虑场景中物体之间相互的反射和透射影响，形成更加真实的图像。

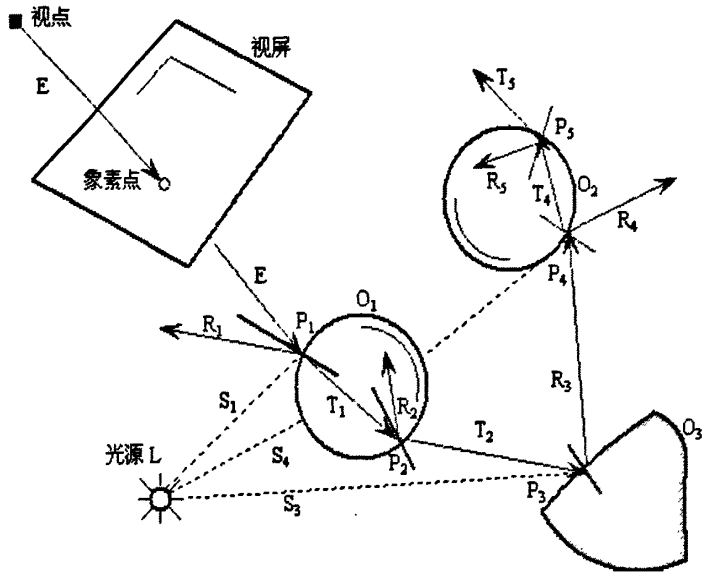
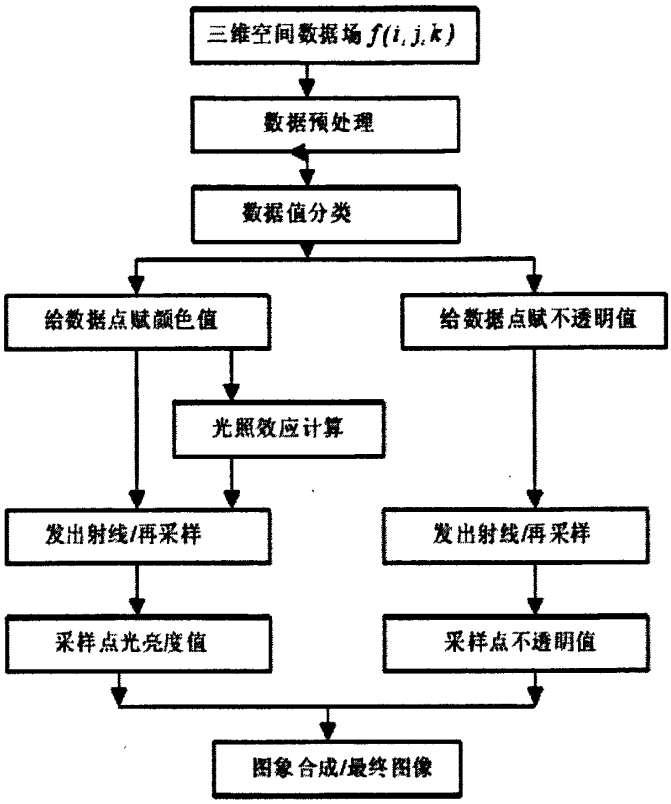


图 3.3 光线跟踪示意图

3.2.2 算法流程图

光线投射体绘制方法的思想是从象平面上每一个象素出发沿视线方向发出一条射线. 这条射线会穿过体素所在的空间，随着射线的传播，把它所遇到的体素的颜色和阻光度进行累积与合成. 当阻光度累积到 1 或射线已经穿过了体素空间时，就停止射线的传播，并把当前合成的颜色作为该象素的颜色写到帧缓存中去.

整个算法流程为:



对于医学上所采集到的体数据 $f(x_i, y_j, z_k)$，$f(x_i, y_j, z_k)$表示均匀分布的三维网格数据点。首先我们要对数据进行预处理，包括数据值的缩放变换，噪点的检测与去除。然后对数据值在以往的经验上进行归类，因为不同器官和组织的数据值是不一样的，所以这一步把它们归来出来，把每一类赋予相应的颜色值和阻光度（不透明值）。这样可以在最终图像里发现轮廓分明的不同组织。接着，我们模拟射线穿过体数据场的过程，在射线上均匀采样，对每一个采样点，用其相邻的附近 8 个点通过三线性插值，得到采样点的颜色值和阻光度。同时加上上面提到的 Phong 模型计算出来的光照值。这样当阻光度累积到 1 或者射线穿过体素场的时候，就结束一次射线的投射，把颜色值保存在计算机里[33][34]。

由于重新采样和图像合成是按屏幕上每条扫描线的每个象素逐个进行的，因该算法又称为图像空间扫描的体绘制算法。通过对每一个像素投射光线，最终得到一副完整的体绘制图[35]。

以下详细描述流程的步骤。

3.2.3 采样点三线性插值

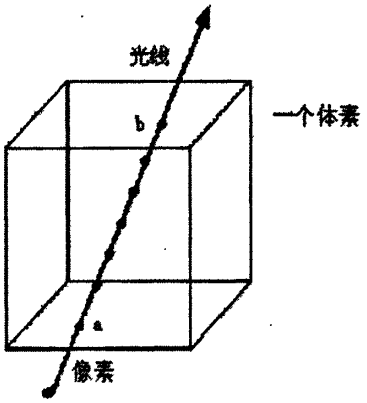
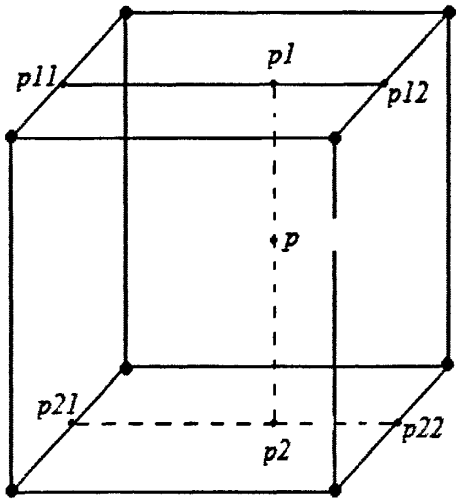


图 3.4 一条射线穿过数据场的一个体素等距采样

一条光线穿过体素场，等距采样的时候，采样点的颜色值和阻光度值是通过三线性插值计算出来的。



设 p 点相对于 p_1 的偏移量为 α，p_1 点相对于 p_{11} 点的偏移量为 β，p_2 点相对于 p_{21} 点的偏移量为 γ。则 p_i 点的颜色值 c_i 可以由三线性插值公式计算:

$$c = c_1 * (1-\alpha) + c_2 * \alpha$$
$$c_1 = c_{11} * (1-\beta) + c_{12} * \beta$$
$$c_2 = c_{21} * (1-\gamma) + c_{22} * \gamma$$

3.2.4 数据点梯度估计

在流程中提到了 Phong 模型计算光照颜色。因为 Phong 模型公式 $I=I_aK_a+I_pK_d(L\bullet N)+I_pK_s(R\bullet V)^n$ 中要用到法向量，但我们的原始数据中并没有法向量的信息。我们可以通过中心差分方法计算梯度逼近数据点的法向量。如下:

$$grad\_x=(f(x_{i-1},y_j,z_k)-f(x_{i+1},y_j,z_k))/2$$

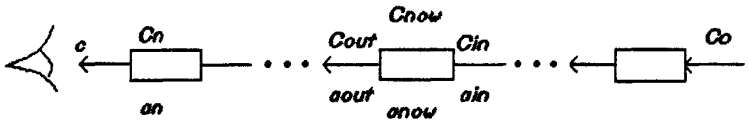
$$grad\_y=(f(x_i,y_{j-1},z_k)-f(x_i,y_{j+1},z_k))/2$$

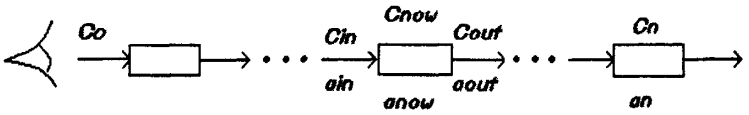
$$grad\_z=(f(x_i,y_j,z_{k-1})-f(x_i,y_j,z_{k+1}))/2$$

由此可以得到每个体素的点的法向量。

3.2.5 图像颜色合成

光线投射从一个像素点出发，投射光线，计算光线经过的采样点的颜色，最终通过颜色累积合成计算此像素点的颜色。计算所有的像素点颜色，则生成了一幅最终的图像。





对于颜色的合成，直观的看来，有两种方法。

1. 由后向前计算累积颜色

这种方法是按照采样点的顺序，由后向前将各个采样点的颜色值和阻光度合成在一起，形成像素点的颜色。如图所示，设第 i 个体素的颜色值为 $c_{now}(c_i)$，阻

光度为 $a_{now}(a_i)$（代表着此像素自身能发挥出多少程度的颜色），进入第 i 个体素的颜色值为 c_{in}，阻光度为 a_{in}，经过第 i 个体素后的颜色值为 c_{out}，阻光度为 a_{out}。则依据由后向前的规则，有如下公式:

$$C_{out}=C_{in}*(1-a_{now})+C_{now}*a_{now}$$

设初始颜色值为 C_0，最终的颜色值为 C，每个体素的透明度为 β_i，则

$$C=C_0\beta_1\cdots\beta_n+C_1\alpha_1\beta_2\cdots\beta_n+\cdots+C_n\alpha_n=C_0\sum_{i=1}^{n}\beta_i+\sum_{i=1}^{n}C_i\alpha_i\sum_{j=i+1}^{n}\beta_j$$

2. 由前向后计算累积颜色

上面的方法必须顺序经过每一个采样点，而采用由前向后的方法，则不必用上每一个采样点的信息。计算公式为

$$C_{out}=C_{in}+C_{now}*(1-\alpha_{in})$$
$$\alpha_{out}=\alpha_{in}+\alpha_{now}*(1-a_{in})$$

由于由前向后的累积颜色，如果不透明度接近了 1，则说明没有必须继续往下计算了（后面的像素点不会贡献颜色），此时的颜色值即为像素点的最终值。此方法可以省去无效的计算，速度比较快，加速体绘制的过程。得到了广泛的采用。

3.3 Shear-Warp 体绘制方法

体绘制有很多种加速绘制方法[36][37][38][39]。1994 年，Philippe Lacroute 和 Marc Levoy 共同发表了《Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation》论文，文中将 Shear-Warp 方法用于体绘制方法，运用错切变换，加速了体绘制的过程。

Shear-Warp 方法将直接体绘制对三维体素场的投影变换分解成:

1. 三维体素场的 Shear 错切变换。

2. 二维图像的 Warp 变形。

经过以上两个步骤，三维空间的重采样过程被转换成为二维平面的重采样过程，大大减少了三维重采样的计算量。有比较高的效率[40]。

Shear-Warp 方法首先将体素场的空间变换到一个被称为“错切对象空间”的

中间坐标系，这是一个过渡坐标系，在这个坐标系中，所有视线均与第三坐标轴平行（主视轴），这使得成像时的物体坐标系能高效地投影到二维图像平面上。从物体空间到错切空间的变换称为错切变换，在进行错切变换后，应保证变换后视线与切片垂直。

由错切变换出来的图像只是中间图像，不是最终图像，还要进行 Warp 变换过程，才形成最终图像。Shear-Warp 算法需要选择一个主视轴，对应三个坐标轴，主视轴应选与视线方向最接近垂直的坐标轴方向[41]。Shear-Warp 算法对于平行投影，只需简单的平移即可实现错切变换，而对于透视投射，在平移的同时还需进行缩放，因为透视投影是有近大远小的视觉效果，所以需要缩放。

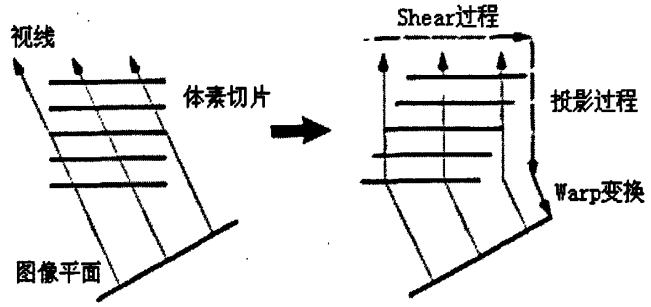


图 3.5 平行投影下错切变换方法的中间坐标变换示意图

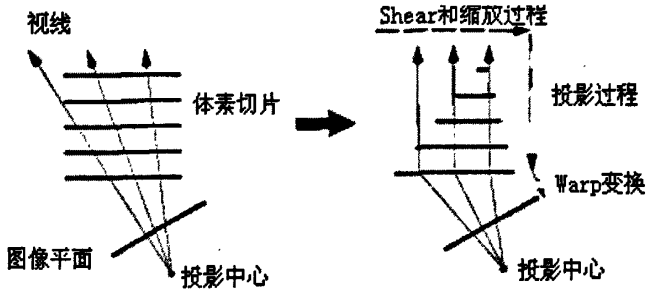


图 3.6 透视投影下错切变换方法的中间坐标变换示意图

Shear-Warp 算法中，假定坐标是按列向量表示，那么视角变换矩阵可以分解为：

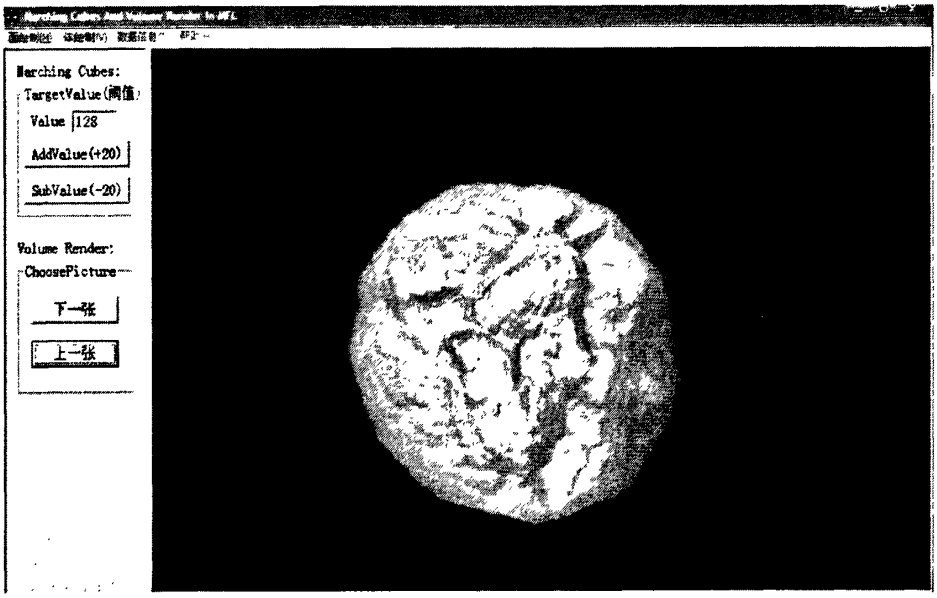
$$M_{view} = M_{warp} \cdot S \cdot P$$

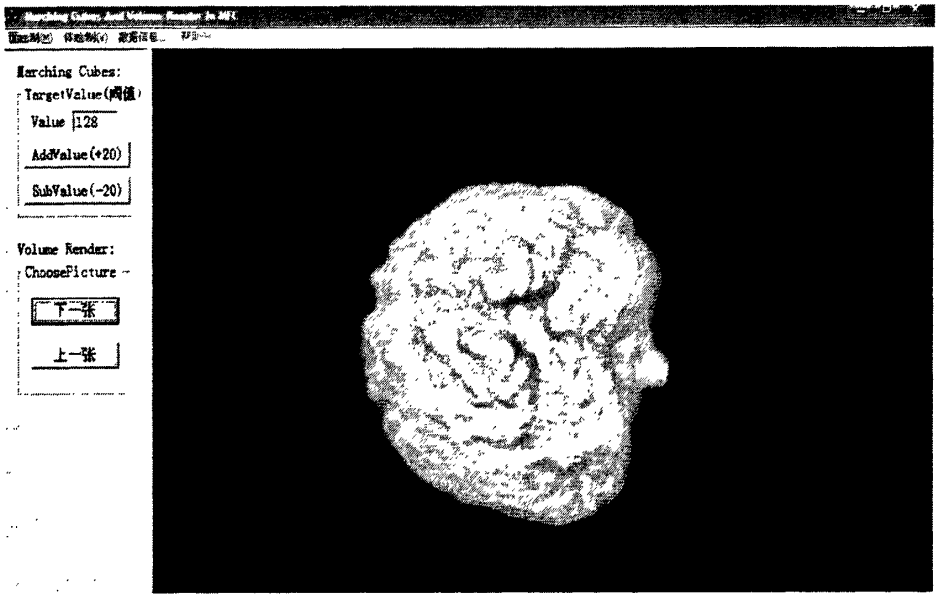
其中 M_{view} 为 Shear-Warp 视角变换矩阵，M_{warp} 为 Warp 变换矩阵，S 为 Shear

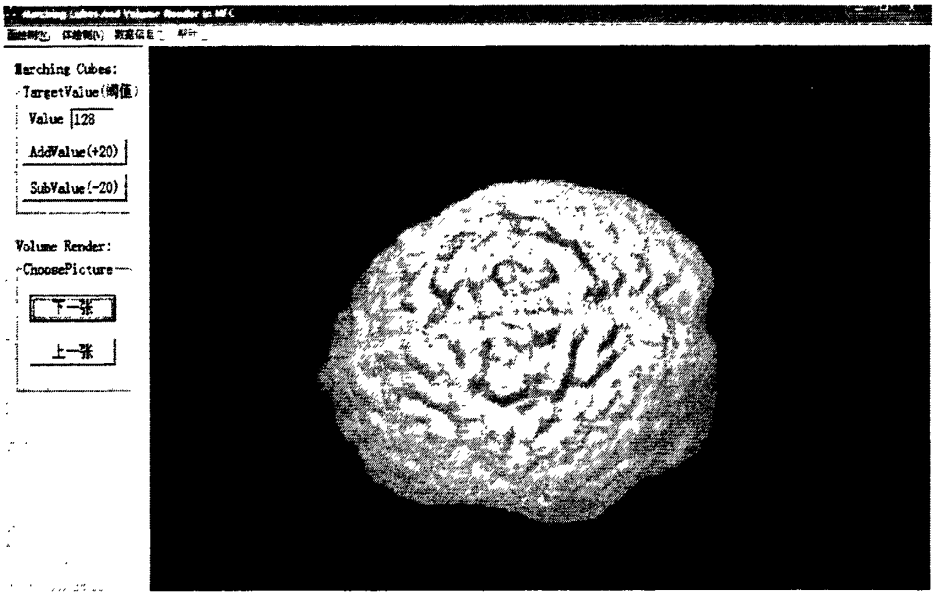
错切矩阵，P是一个转轴矩阵，让 z 轴变为主视角轴。通过分解为三维数据场的错切变换和二维图像的变形两步来进行三维空间的重采样过程转换为二维平面的重采样过程，大大减少了投影过程的计算量。

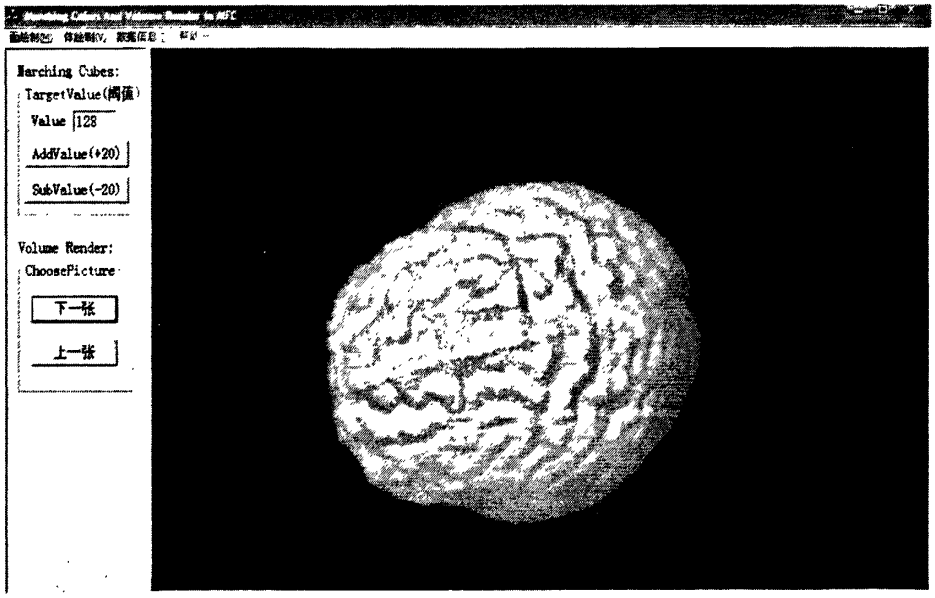
3.4 体绘制算法实现示例

由于体绘制不同于面绘制，它是按照光线投射进而生成图像，只能通过生成多幅图像来达到旋转的目的。以下是体绘制方法生成的 50 张图像中的 4 张，这 50 张图像是按固定角度（可以设置，默认为 5 度）旋转所形成的一系列的图像。

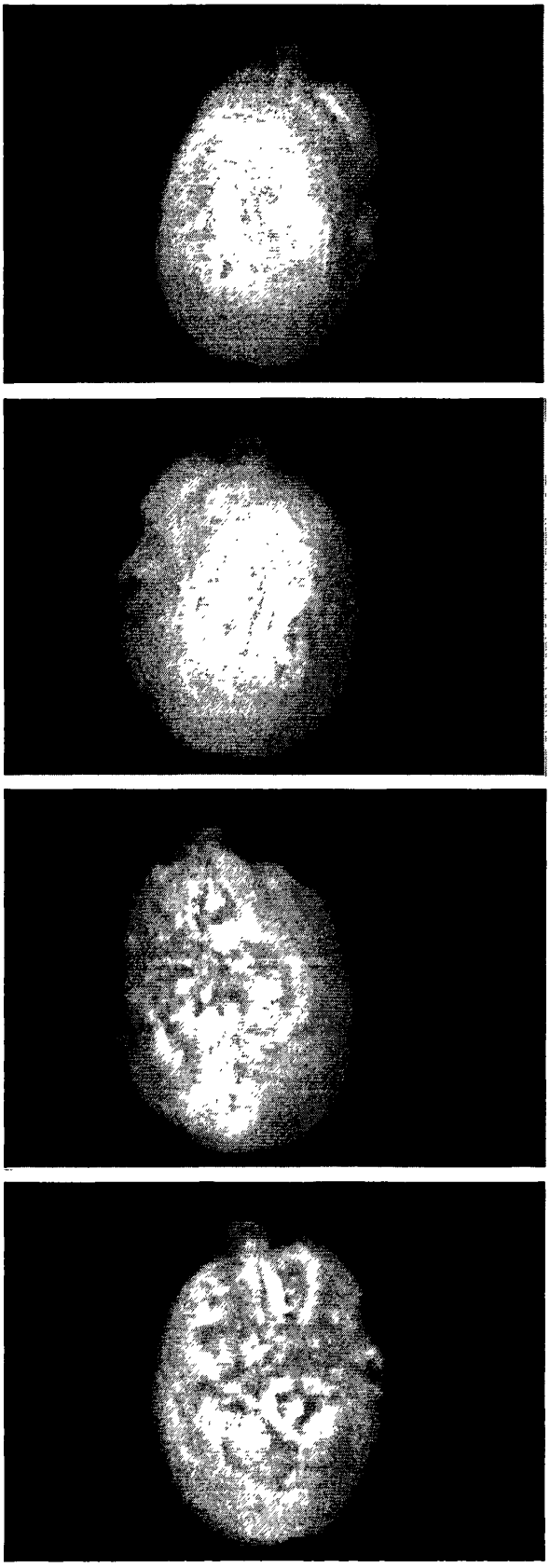








同大脑三维数据一样，第二章的人体头部的数据也可以体绘制展现出来。但是由于此三维数据的原始维度比较小，所以生成的图像质量不是很好，分辨率比较低。



第四章 基于GPU的三维重建算法软件平台

在计算机中显示图形是一个很有趣的事情。C语言提供了graphics图形库，很多经典的小游戏，都可以用C语言进行编写，以图形的方式展现给用户。但是如果需要写比较复杂的程序，会导致程序非常庞大，开发效率低下，可移植性差。

4.1 OpenGL介绍

OpenGL（全写 Open Graphics Library）是一个定义了一个跨编程语言、跨平台的编程接口的规格，它用于三维图象（二维的亦可）。OpenGL是个专业的图形程序接口，是一个功能强大，调用方便的底层图形库[42][43]。

OpenGL 是行业领域中最为广泛接纳的 2D/3D 图形 API，其自诞生至今已催生了各种计算机平台及设备上的数千优秀应用程序。OpenGL 是独立于视窗操作系统或其它操作系统的，亦是网络透明的。在包含CAD、内容创作、能源、娱乐、游戏开发、制造业、制药业及虚拟现实等行业领域中，OpenGL 帮助程序员实现在 PC、工作站、超级计算机等硬件设备上的高性能、极具冲击力的高视觉表现力图形处理软件的开发。

4.1.1 OpenGL特点及功能

OpenGL是一个开放的三维图形软件包，它独立于窗口系统和操作系统，以它为基础开发的应用程序可以十分方便地在各种平台间移植；OpenGL可以Visual C++紧密结合，便于实现机械手的有关计算和图形算法，可保证算法的正确性和可靠性；OpenGL使用简便，效率高。它具有七大功能：

1.建模：OpenGL 图形库除了提供基本的点、线、多边形的绘制函数外，还提供了复杂的三维物体（球、锥、多面体、茶壶等）以及复杂曲线和曲面绘制函数。

2.变换：OpenGL图形库的变换包括基本变换和投影变换。基本变换有平移、旋转、变比镜像四种变换，投影变换有平行投影（又称正射投影）和透视投影两种变换。其变换方法有利于减少算法的运行时间，提高三维图形的显示速度。

3.颜色模式设置：OpenGL颜色模式有两种，即RGBA模式和颜色索引（Color

Index)。

4.光照和材质设置：OpenGL光有辐射光（Emitted Light）、环境光（Ambient Light）、漫反射光（Diffuse Light）和镜面光（Specular Light）。

5:纹理映射（Texture Mapping）。利用OpenGL纹理映射功能可以十分逼真地表达物体表面细节。

6:位图显示和图象增强图象功能除了基本的拷贝和像素读写外，还提供融合（Blending）、反走样（Antialiasing）和雾（fog）的特殊图象效果处理。

7：双缓存动画（Double Buffering）双缓存即前台缓存和后台缓存，简言之，后台缓存计算场景、生成画面，前台缓存显示后台缓存已画好的画面。

此外，利用OpenGL还能实现深度暗示（Depth Cue）、运动模糊（Motion Blur）等特殊效果。从而实现了消隐算法[44]。

4.1.2 OpenGL硬件加速

在图形加速卡没有发明之前，OpenGL都是使用的软加速，即所有的图形渲染工作最终都是由CPU来完成的。自从图形加速卡（现在称显卡）的诞生，大大加快了图形的显示。显卡厂商都积极支持OpenGL标准，比如Nvidia，AMD的显卡，都支持OpenGL。我们调用OpenGL的API函数的时候，终于调用到了显卡的驱动程序，渲染工作交由显卡来完成，最终完成图形显示的加速。所有在有显卡的计算机中即可实现基于GPU加速的医学三维影像算法。

4.2 OpenGL的安装

在计算机中安装OpenGL很简单。在安装了Visual Studio之后，gl.h，glu.h,glu32.dll,opengl32.dll就会安装到相应的目录。如果没有安装Visual Studio集成开发环境，只需要把对应的h文件，lib文件，dll文件拷贝到相应的目录（include目录）即可在程序中使用。

4.3 Visual C++中的MFC结合OpenGL建立三维绘图框架

在MFC中使用OpenGL，需要按照一定的步骤进行初始化。按照微软MSDN的

介绍，Windows提供了一系列专有的API来进行OpenGL。

下面简要介绍一下Win32下使用OpenGL函数库特殊的初始化过程。首先，重新设置画图窗口的象素格式，使其符合OpenGL对象素格式的需要。为此需声明一个PIX-ELFORMATDESCRIPTOR结构的变量，并适当地设置某些结构成员的值，使其支持OpenGL及其颜色模式。变量的声明见后面SetupPixelFormat()函数的描述。再以此变量为参数调用ChoosePixelFormat()函数分配1个象素格式号，然后调用 SetPixelFormat()将其设置为当前象素格式。完成了象素格式的重新设置后，需要为OpenGL建立着色上下文(RenderContext)。着色上下文的作用类似于Windows设备上下文DC。只有建立了着色上下文RC后，OpenGL才能调用绘图原语在窗口中做出图形。Win32 API提供了几个操作着色上下文的函数，包括建立、复制、使用、删除、查询等，它们都以wgl为词头。着色上下文是以线程为单位的，每一个线程必须使用 1 个着色上下文作为当前着色上下文才能执行OpenGL绘图原语。wglCreateContext()是建立着色上下文的函数，它以1个设备上下文句柄为参数，返回1个与此设备上下文相联的着色上下文句柄。再以此句柄为参数调用函数 wglMakeCurrent()使着色上下文成为线程当前使用的着色上下文，完成Windows下OpenGL绘图环境的初始化过程。

本文所建立的框架正是基于以上的思想。在用MFC生成一个最基本的单文档程序之后。需要增加如下代码。

1.建立像素格式

```
//Initial OpenGL in MFC (1)SetWindowPixelFormat
BOOL CMarchingCubesView::SetWindowPixelFormat(HDC hDC)
{
    PIXELFORMATDESCRIPTOR pixelDesc;
    pixelDesc.nSize = sizeof(PIXELFORMATDESCRIPTOR);
    pixelDesc.nVersion = 1;
    pixelDesc.dwFlags =
          PFD_DRAW_TO_WINDOW
        | PFD_DRAW_TO_BITMAP
```

```
    | PFD_SUPPORT_OPENGL
    | PFD_SUPPORT_GDI
    | PFD_STEREO_DONTCARE;

pixelDesc.iPixelType = PFD_TYPE_RGBA;
pixelDesc.cColorBits = 32;
pixelDesc.cRedBits = 8;
pixelDesc.cRedShift = 16;
pixelDesc.cGreenBits = 8;
pixelDesc.cGreenShift = 8;
pixelDesc.cBlueBits = 8;
pixelDesc.cBlueShift = 0;
pixelDesc.cAlphaBits = 0;
pixelDesc.cAlphaShift = 0;
pixelDesc.cAccumBits= 64;
pixelDesc.cAccumRedBits = 16;
pixelDesc.cAccumGreenBits = 16;
pixelDesc.cAccumBlueBits = 16;
pixelDesc.cAccumAlphaBits= 0;
pixelDesc.cDepthBits = 32;
pixelDesc.cStencilBits= 8;
pixelDesc.cAuxBuffers = 0;
pixelDesc.iLayerType= PFD_MAIN_PLANE;
pixelDesc.bReserved = 0;
pixelDesc.dwLayerMask= 0;
pixelDesc.dwVisibleMask= 0;
pixelDesc.dwDamageMask= 0;
m_GLPixelIndex = ChoosePixelFormat( hDC, &pixelDesc);
```

```
    if (m_GLPixelIndex==0) // Let's choose a default index.
    {
        m_GLPixelIndex = 1;
        if (DescribePixelFormat(hDC, m_GLPixelIndex,
            sizeof(PIXELFORMATDESCRIPTOR), &pixelDesc)==0)
        {
            return FALSE;
        }
    }

    if (SetPixelFormat( hDC, m_GLPixelIndex, &pixelDesc)==FALSE)
    {
        return FALSE;
    }

    return TRUE;
}
```

2.建立着色上下文，并与MFC的上下文进行关联

```
//Initial OpenGl in MFC (2)CreateViewGLContext
BOOL CMarchingCubesView::CreateViewGLContext(HDC hDC)
{
    m_hGLContext = wglCreateContext(hDC);
    if (m_hGLContext == NULL)
    {
        return FALSE;
    }
    if (wglMakeCurrent(hDC, m_hGLContext)==FALSE)
    {
        return FALSE;
```

```
    }
    return TRUE;
}
```

3.初始化OpenGL

```
//Initial OpenGl in MFC (3) put the two initial funcitons into myInit()
int CMarchingCubesView::myInit()
{
    HWND hWnd = GetSafeHwnd();
    HDC hDC = ::GetDC(hWnd);
    if (SetWindowPixelFormat(hDC)==FALSE)
        return 0;
    if (CreateViewGLContext(hDC)==FALSE)
        return 0;

    //add marching cubes init func here in myinit
    MCInit();

    return 1;
}
```

4.在程序结束之后，把上下文关联取消

```
//Destroy OpenGL when View is Destroying
void CMarchingCubesView::OnDestroy()
{

    // TODO: Add your message handler code here
    if(wglGetCurrentContext()!=NULL)
    {
        wglMakeCurrent(NULL, NULL) ;
    }
```

```
    if (m_hGLContext!=NULL)
    {
        wglDeleteContext(m_hGLContext);
        m_hGLContext = NULL;
    }
    CView::OnDestroy();
}
```

4.保证OpenGL的视窗与MFC的视窗同步

```
//Reshape
void CMarchingCubesView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // TODO: Add your message handler code here

    GLsizei width, height;
    width = cx; height = cy;

    if (cy>0)
    {
        ArcBall.setBounds(width,height);
        /*GLfloat          fAspect,          fHalfWorldSize          =
(1.41421356237309504880168872420 97/2); */
        GLfloat fAspect, fHalfWorldSize = 0.6;

        glViewport( 0, 0, width, height );
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
```

```
        if(width <= height)
        {
            fAspect = (GLfloat)height / (GLfloat)width;
            glOrtho(-fHalfWorldSize,                    fHalfWorldSize,
-fHalfWorldSize*fAspect,
                fHalfWorldSize*fAspect,              -10*fHalfWorldSize,
10*fHalfWorldSize);
        }
        else
        {
            fAspect = (GLfloat)width / (GLfloat)height;
            glOrtho(-fHalfWorldSize*fAspect,    fHalfWorldSize*fAspect,
-fHalfWorldSize,
                fHalfWorldSize, -10*fHalfWorldSize, 10*fHalfWorldSize);
        }

        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();
    }

}
```

4.4 本文程序的架构

本文是基于MFC结合OpenGL来实现医学三维影像重建。有2个类来分别实现面绘制和体绘制.

```
class CMarchingRender
{
public:
```

```
    CMarchingRender();
    virtual ~CMarchingRender();
    ...
};
```

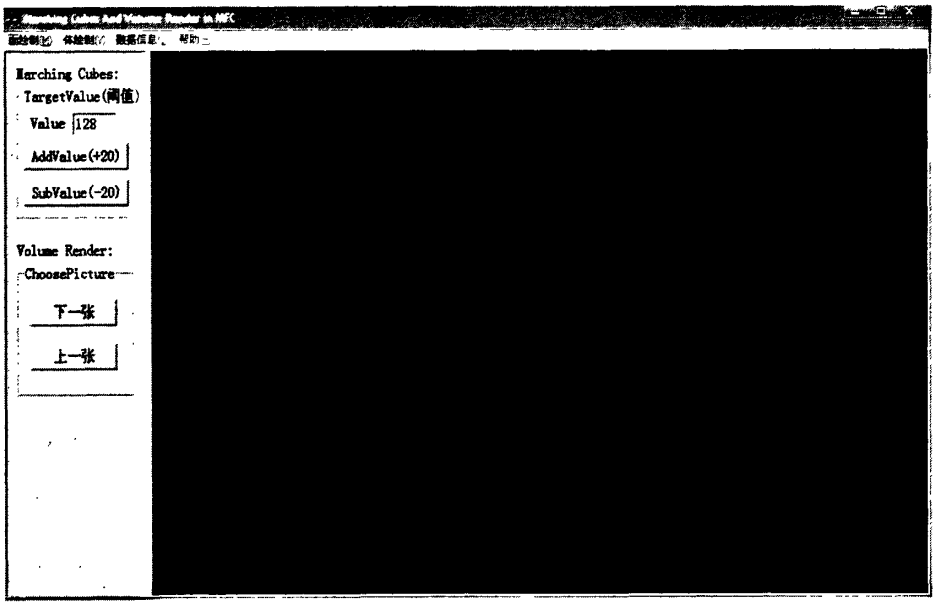
CMarchingRender是实现Marching Cubes方法的抽象出来的类，它封装了全部用于实现算法的函数接口，在MFC主视窗类中定义这个类，并调用相应的方法，即可进行MC算法的三维重构。

在视类中定义一个CMarchingRender的变量marRender，通过调用vReadDimData函数接口来载入数据的三维信息，vReadRenderData函数接口读取数据值。marchCubes处理某一个体素，生成三角面片。

```
class CVolumeRender
{
public:
    CVolumeRender();
    virtual ~CVolumeRender();
...
};
```

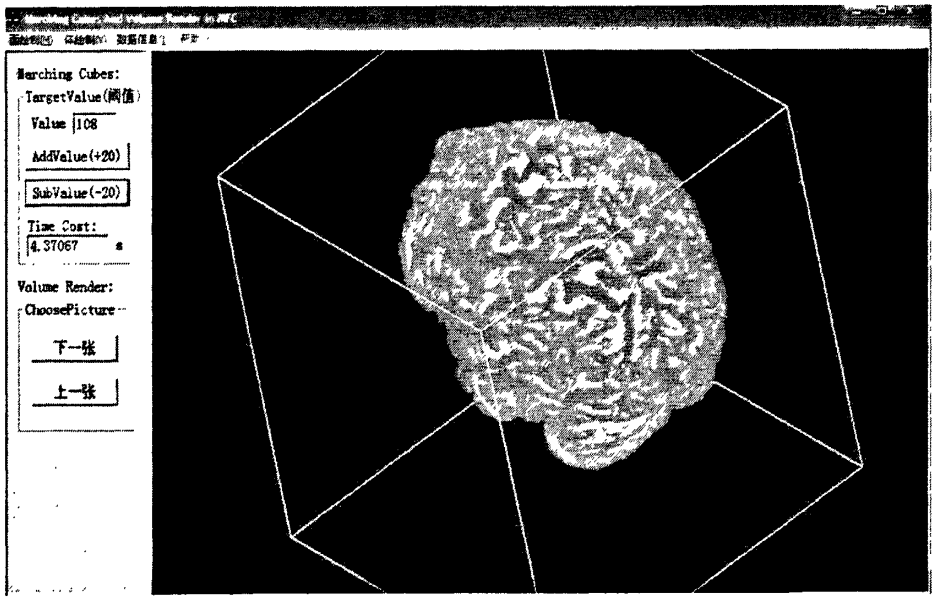
CVolumeRender 是实现直接体绘制方法的抽象出来的类，它封装了全部用于实现算法的函数接口，在MFC主视窗类中定义这个类，并调用相应的方法，即可进行体绘制算法的三维重构。

在视类中定义一个CVolumeRender的变量volRender，通过调用vReadDimData函数接口来载入数据的三维信息，render函数接口进行体绘制。

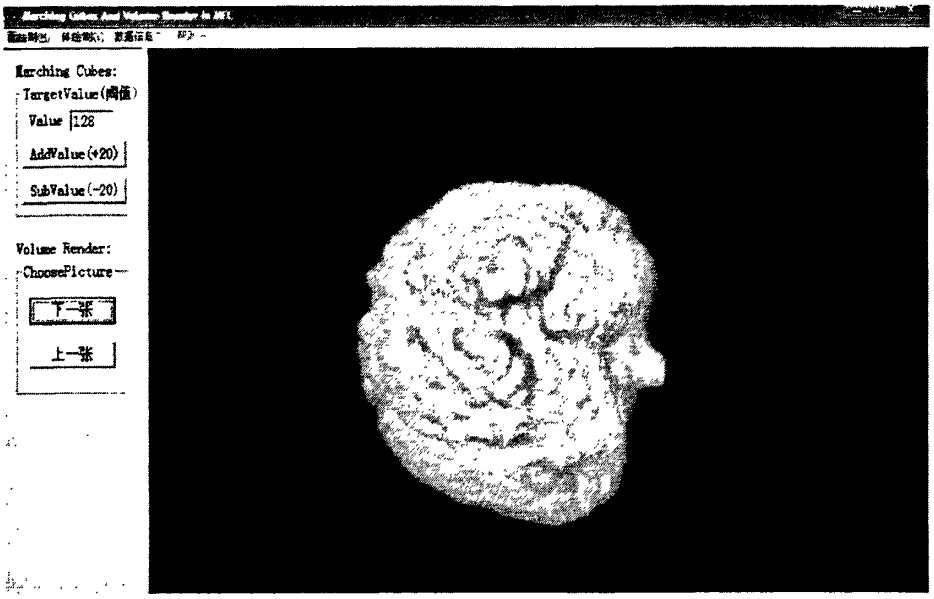


上图是程序框架的运行截图，只要点击对应的菜单项，载入数据，调用绘制方法，同时可以在左边的控制台中定义一些操作，于是就可以在右边的黑色视窗中绘制出三维结果。

下图为面绘制的一张截图:



下图为体绘制的一张截图:


Marching Cubes:
TargetValue(阈值)
Value 128
AddValue(+20)
SubValue(-20)
Volume Render:
ChoosePicture
下一张
上一张

第五章 总结

医学影像可视化是医学和计算机科学计算可视化研究领域的一个重要分支，它涉及到计算机图形学、数字图像处理、生物医学工程等多种技术，是一项多学科交叉的研究课题。它把计算机三维图形学技术融入医学工程，实现了医学的突破。虽然医学影像三维重建已经有了二十多年的研究和进展，但其仍是热门的研究领域。

本文综合了许多文献资料，详细介绍了医学三维影像可视化的方法，对面绘制和体绘制进行了详细的阐述。

在面绘制 Marching Cubes 方法上，本文不仅在计算机中实现了算法，同时提出了一个基于棱边共享等值点的改进，对算法进行了优化改进，使等值点的计算量减少了将近一半，整个算法的时间降低了 20%左右。本文还在此基础上，将多线程并行计算运用于算法的实现，使得同时有 2 个线程进行 Marching Cubes 算法处理，在棱边共享等值点的改进基础上，整个算法时间又降低了 20%左右。本文还介绍了三维图形开发包 OpenGL，整个程序开发都是在 Visual C++ 6.0 的基础上，结合 OpenGL，以 MFC 为框架完成的。面绘制重建出来的三维图形可以任意角度旋转，可以随意设置阈值，观看不同的等值面。

对于论文的进一步完善和改进，我个人认为有以下几点:

对于面绘制算法，可以在本文改进的基础上，对所生成的三角面片进行一定的网格简化，以加速重建的速度，当发现问题时，再局部非简化精确重建，这样能更快的找到病灶，对症下药，节省时间。

对于医学三维影像可视化，系统的实时性要求很强。基于云计算分布式计算平台的应用，比如 Hadoop 分布式系统，就可以应用到面绘制，发挥其原生支持分布式并行的特点，大大增强结果的精确性和实时性。

在面绘制的基础上，进一步研究虚拟漫游技术，让视角在三维重建的等值面中变化游走，彻底观察医学图像，提供更加准确的判断。

医学影像三维重建在我国还是非常新兴的研究方向，近几年得到了飞速的发展，相信通过我国研究者的不断努力，我国的医学三维技术一定会为医疗诊断提供强有力的工具，让科技造福于人民。

参考文献

[1] 唐泽圣等. 三维数据场可视化. 北京: 清华大学出版社. 1999 年 12 月.

[2] B. S. Kuszyk. D. R. Ney, etal, The current state of the art in three dimensional oncologic imaging: an overview, I. J. Radiation Oncology Bio. Phy., 1995, 33(5): 1029-1039.

[3] 石教英等. 科学计算可视化算法与系统. 科学出版社, 1996-09.

[4] B. H. McCormick, A. T. DeFanti, and M. D. Brown. Visualization in Scientific Computing. Computer Graphics, 21(6), 1987.

[5] Stefan Neubauer. MRI and CT. Medicine, Volume 34, Issue 4, 1 April 2006, Pages 157-161.

[6] 鲍华; 基于直接体绘制技术的医学图像三维可视化方法研究[D]; 四川大学; 2006 年

[7] Duurst M J. Additionalreference to marching cubes. Computer Graphics, 1988, 22(2): 72 - 73.

[8] Hohne, K. H. and Bernstein, R. Shading 3D-Images from CT Using Gray-Level Gradients. IEEE Trans. on Medical Imaging MI-5, 1 (March 1986), 45-47.

[9] Gordon, D. and Reynolds, R. A. Image Space Shading of 3-Dimensional Objects. Computer Graphics and Image Processing 29, 3 (March 1985), 361-376.

[10] Manohar S. Advanced in volume graphics. Computerized&Graphics, 1999, Vol: 23, pp. 73-84.

[11] Marc Levoy. Efficient Ray Tracing of Volume Data. ACM Transactions on Graphics 1990, 9(3): 245 — 261

[12] Yagel R, Machiraju R. Data-parallel volume rendering algorithms. The Visual Computer, 1995, 11(6): 319-338

[13] C. Silva and J. Mitchell, "The Lazy Sweep Ray Cast-ing Algorithm for Rendering Irregular Grids," IEEE Transactions on Visualization and

Computer Graphics, 1997, 3(2): 142-157.

[14]M. W. Jones and M. Chen. A new approach to the construction of surfaces from contour data. Computer Graphics Forum, 1994, 13(3): 75-84.

[15]Levoy, Marc. Display of surfaces from volume data. IEEE Computer Graphics & Applications, 8(3): 29-37, May 1988

[16]D. R. Ney, E. K. Fishman. Three dimensional imaging of CT: techniques and applications. Proceeding of Visualization in Biomedical Computing, 1990, 498-506.

[17]Levoy, Marc. Volume rendering by adaptive refinement. The Visual Computer, 6(1): 2 - 7, February 1990.

[18]Timothy S. Newman, Hong Yi. A survey of the marching cubes algorithm. Computers & Graphics, Volume 30, Issue 5, October 2006, Pages 854-879.

[19]K. S. Delibasis, G. K. Matsopoulos, N. A. Mouravliansky, K. S. Nikita. A novel and efficient implementation of the marching cubes algorithm. Computerized Medical Imaging and Graphics, Volume 25, Issue 4, July 2001, Pages 343-352.

[20]Seungtaik Oh, Bon Ki Koo. Data perturbation for fewer triangles in marching tetrahedra. Graphical Models, Volume 69, Issues 3-4, May-July 2007, Pages 211-218.

[21]Lorensen W, Chine HE. Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics, 1987. 21(4) 163-169.

[22]Lacroute P, Levoy M. Fast volume rendering using a shearwarp factorization of the viewing transformation. Proceedings of ACM SIGGRAPH' 94[C]. New York: ACM Press, 1994. 451-458.

[23]M. J. Durst. Letters: Additional Reference to "Marching Cubes". Computer Graphics, Vol. 22(2). 1988.

[24]Ira J. Kalet, Robert S. Giansiracusa, Jonathan Jacky, Drora Avitan. A declarative implementation of the DICOM-3 network protocol. Journal of Biomedical Informatics, Volume 36, Issue 3, June 2003, Pages 159-176.

[25]S. van Foreest-Timp. Iso-surface volume rendering for implant surgery. International Congress Series, Volume 1230, June 2001, Pages 733-738.

[26]宋海友，蒲立新，医学可视化中三维重建的基本算法，中国医院，2005.11，增刊第九卷。

[27]Nielson G.M, Hamann B. The asymptotic decider: resolving the ambiguity in marching cubes. IEEE Proceedings of Visualization' 91, 83～91.

[28]Doi A, Koide A. An Efficient Method of Triangulating Equi-Valued Surfaces by sing Tetrahedral Cells. IEICE Transactions, E74(1), 214-224.

[29]Drebin R A, Carpenter L, Hanrahan P. Volume Rendering. Computer Graphics, 1988. 22(4), 65-74.

[30]Levy M. Volume Rending by Adaptive Refinement. UNC Technical Report, 1988, June, 88030.

[31]吴庆标，韩丹夫. 计算机图形学. 浙江大学出版社. 2006. 6.

[32]Levy M. Cff'icient Ray Tracing of Volume Data. ACM Transactions on Graphics, 1990. 9(3), 245-261.

[33]Jae Jeong Choi, Byeong-Seok Shin, Yeong Gil Shin, Kevin Cleary. Efficient volumetric ray casting for isosurface rendering Computers & Graphics, Volume 24, Issue 5, October 2000, Pages 661-670.

[34]Sudhanshu K. Semwal, Brian K. Barnhart. Ray casting and the enclosing-net algorithm for extracting shapes from volume data Computers in Biology and Medicine, Volume 25, Issue2, March 1995, Pages 261-276.

[35]王文成等. 加速体绘制技术. 计算机辅助设计与图形学学报, 2002-09.

[36]Ren Ken. Ruei-Chuan Chang, Ray-cast volume rendering accelerated by incrental trilinera interpolation and cell templastes[J], the Visual Computer, 1995, 11(6): 297-308.

[37]J Comba, J T K Iosowski, N Max, etal. Fast Polyhedral cell sorting for interactive rending of unstrured grids[J]. Computer Graphics Forum, 1999, 18(3), 369-376.

[38]Gelder A.V.,Kim K.Direct volume rendering with shading via 3D textures[A],Proceedings of ACM Symposium on Volume Visualization, San Francisco,ACM Press,1996,pp.23-30.

[39]Ghosh A,Prabhu P,Kaufman A.E.,Mueller K.Hardware assisted multichannel volume rendering.Proceedings of the International Conference on Computer Graphics,July 9-11,2003,pp.1-6.

[40]Ana Elisa F. Schmidt, Marcelo Gattass, Paulo Cezar P. Carvalho.Combined 3D visualization of volume data and polygonal models using a Shear-Warp algorithm Computers & Graphics, Volume 24, Issue 4, August 2000, Pages 583-601.

[41]Jürgen P. Schulze, Ulrich Lang.The parallelized perspective shear-warp algorithm for volume rendering.Parallel Computing, Volume 29, Issue 3, March 2003, Pages 339-354.

[42]Tom Mcreynolds, David Blythe.OpenGL implementations Advanced Graphics Programming Using OpenGL, 2005, Pages 129-151.

[43]OpenGL 体系结构审核委员会 著，邓郑祥 译.OpenGL 编程指南（第四版）.人民邮电出版社.2005.04.

[44]和平鸽工作室.OpenGL 三维图形系统开发与实用技术.清华大学出版社,2003-08.