山东大学

毕 业 论 文(设 计)

论文（设计）题目:

蝙蝠 CT 图像的分割和三维表面模型重建

姓　　名 陈建锋

学　　号 201000301010

学　　院 软件学院

专　　业 软件工程

年　　级 2010 级

指导教师 潘荣江

2014 年 5 月 21 日

目　录

蝙蝠 CT 图像的分割和三维表面模型重建

摘 要

本课题由蝙蝠肢部骨骼 CT 显微切片出发，建立从分析，分割，合成到三维建模的模型.

本文首先对仿生声学及蝙蝠这一背景作简单的介绍；然后开始描述实验的原始数据，介绍切片图像的特征.

接着是图像预处理. 这一章主要介绍在对CT 切片进行分割之前需要进行的一些预处理操作，例如去除扫描过程中的，无关干扰物对应的图像，对骨骼图像进行锐化处理等.

图像分割一章中,首先简要介绍进行分割的目的，接着讨论分割的方法,最后将介绍实验的过程及效果.

下一部分--合成三维模型. 该过程主要使用VTK库函数对经前面处理过的单张模型进行合成. 文章会给出库函数的简介，合成的方法和合成效果.

最后一步是三维表面模型建模. 3D 表面模型建模是本课题的第二个重要组成部分. 本文将给出建模的目的，接着介绍LS Meshes 建模方法，最后将这个方法运用到本课题中，并对最终的结果进行讨论.

这是一个基于实际问题的医学图像处理问题，因此所有的方法和模型均是基于课题的特定问题出发并做出优化，适用于相类似的问题的建模.

关键字：医学图像处理；直方图匹配；VTK；LS Meshes

ABSTRACT

This project begins with the original CT images of bats' limbs, try to construct a complete model from data analysis, segmentation to 3D composition.

Firstly, we gave a brief introduction to the research background and original images. Also, we would give an analysis to these images.

Then, we did some prepossessing before the segmentation, such as eliminating redundant noises, sharpening the marginal for skeleton, etc.

Thirdly, we discussed how the get the segmentation for each skeleton.

Next, we composed the single 2D images into three dimensional model. This is the first important part for this project. This step was done with the help of VTK classes.

The last part is 3D model reconstruction. In this thesis we gave the purpose of reconstruction, introduced the Least squares meshes method and applied it into our project.

This is a medical image processing specific problem. Consequently, all the method and model are optimized so that it can adjust to THIS problem. Certainly, it can be used in the similar problems.

Keywords: Medical Image Processing; Histogram matching; VTK; LS Meshes

第 1 章　综述

1.1　仿生声学与蝙蝠

仿生学作为一门交叉学科, 它的产生可以回溯到上世纪 50 年代末, 当时在美国俄亥俄州 Wright-Patterson 空军基地工作从事研究工作的 J. E. Steele 首次提出了仿生学 (Bionics) 这一概念. 他认为仿生学就是研究以模仿生物系统、以具有生物系统特征的方式、或是以类似于生物系统方式工作的系统的科学 [1]. 从字面上理解,Bionics 由 “bio” 和 “nic” 构成, 其中 “bio” 在希腊文即意为 “生命”,“nic” 则有 “具有……性质” 的意思.

仿生声学为仿生学的一个研究方向, 它通过对蝙蝠等通过声波衍射进行定位的生物的研究, 获知其中的相关机理, 从而将其模仿并运用于军事、民用等领域, 如改良声纳, 制作人造耳等.

蝙蝠具有极强的声音分辨能力, 其耳内具有生物波的定位结构; 它是目前所知惟一能真正飞行的哺乳动物, 十分适合在黑暗环境中生活. 虽然它们的眼睛几乎不起作用, 但它们可以通过发射生物波并根据其反射的回音辨别物体. 飞行的时候由口和鼻发出一种人类听不到的生物波, 遇到阻碍物或昆虫后会反弹回来. 蝙蝠用耳朵接收后, 就会知道阻碍物或猎物的具体位置, 从而前往捕捉. 它能听到的声音频率可达 300kHz/s, 而人类的一般在 14kHz/s 以下. 因而蝙蝠是仿生声学的最佳研究对象.

1.2　课题研究背景及意义

本研究课题的主题是 “蝙蝠 CT 图像的分割和三维表面模型重建”, 即要对由计算机断层显微扫描而得来的蝙蝠局部骨骼图像进行处理. 整个课题的具体

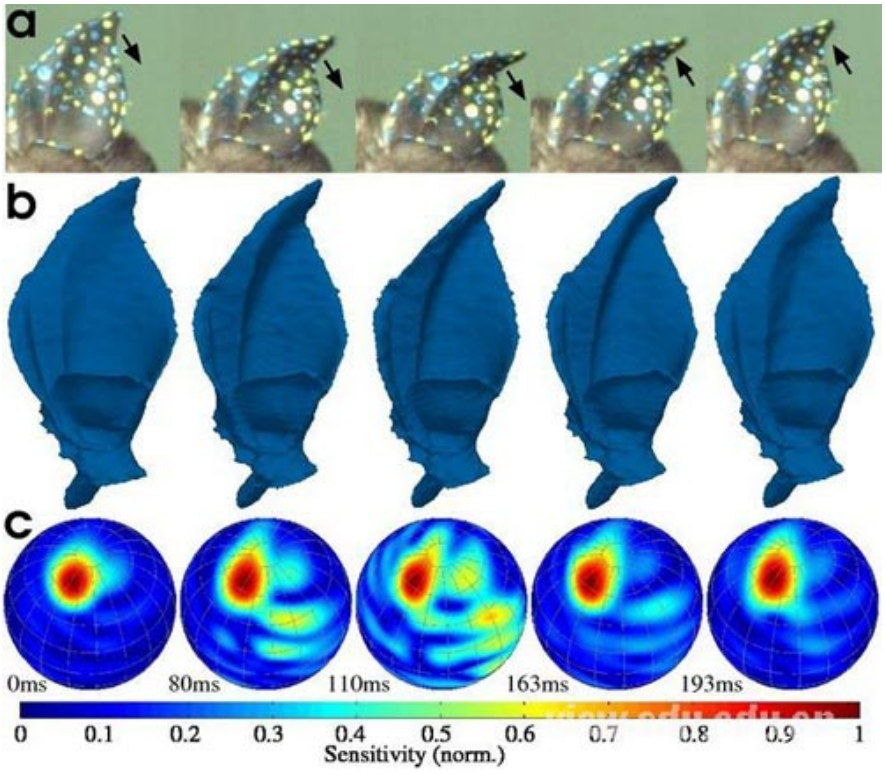


图 1.1 论文 [2] 中插图, 图示为耳朵形变过程中的照片, 此为一个形变周期, 中间照片形变最大

要求为:1) 能够对 CT 扫描而得来的图像进行预处理;2) 设计适用于这个问题的、图片数据较大的图像分割算法;3) 对其骨骼进行三维建模, 最后输出保存为 ply 格式, 且模型的进度要尽可能的高.

作为医学图像处理的一个具体应用, 本课题要求对某一个特定的图像 –蝙蝠骨骼的 CT 显微图像进行处理. 这就使得我不能仅仅使用通用的图像处理方法对图像进行处理和分割. 同时, 出于对后续课题研究的需求, 我还要将这些 CT 切片合成成三维图形. 并且要找到一种合适的方法对其进行三维表面模型重建. 由于医学图像有数据量大, 精度要求高等特点, 本课题也要试图寻求一种合适合理的方法进行建模.

本课题是山东大学仿生声学研究项目的一个组成部分, 也是很基础的部分. 由于这个研究项目需要分析蝙蝠利用生物波进行定位的具体机理, 所以有很多的计算机模拟以及生物实验. 而这些实验的基础就是要知道蝙蝠骨骼的构成. 为

探究蝙蝠骨骼的构成, 我们首先利用计算机断层扫描得到了其局部骨骼的显微图像. 当进行本课题所述的处理之后就会得到一个利于计算机储存, 编程以及模拟分析的三维图形模型. 鉴于此, 本课题是这个研究项目必不可少的一个组成部分.

1.3 课题研究过程

本课题从计划到完成为四个月左右的时间 (2014 年 2 月到 2014 年 5 月), 整个课题分为以下几个步骤:

i. 首先对二维 CT 切片进行预处理：比如去除边界中的系统信息，把用于骨骼之外无关的支撑物 (如玻璃容器、支架等) 对应的图像删除，去除骨腔中的填充物对应的图像等;

ii. 接着使用锐化滤波器突出其骨骼的边缘;

iii. 完成上面过程后对图像进行分割, 以使每一块骨骼独立出来;

iv. 分割完成后将 CT 图像合成成 ply 格式文件。与此同时，尝试在进行二维图像预处理后直接合成成 ply 格式文件，使用 meshlab 等开源工具对其进行分割;

v. 最后一步，使用 least-squares mesh 等方法对三维模型进行建模，得到精度高，且数据量小的模型.

具体的时间安排及论文写作计划如表 [1.1].

表 1.1 课题时间表

开始时间	结束时间	具体工作
2014-02-20	2014-02-28	确定研究过程及具体计划
2014-03-01	2014-03-20	配置学习使用 VTK,meshlab 等软件
2014-03-21	2014-04-10	完成图像预处理及初步分割, 着手完成毕业论文正文
2014-04-12	- - -	中期检查
2014-04-11	2014-04-30	修正分割, 合成三维模型
2014-05-01	2014-05-10	使用 LS mesh 方法生成进度更高的模型，继续完成毕业论文
2014-05-11	2014-05-20	完成及完善毕业论文
- - -	2014-05-30	毕业答辩

1.4 本文的内容安排

在前面的几节中, 首先介绍了仿生声学与蝙蝠, 接着介绍了这个课题的研究主题及其背景意义, 最后阐述了本课题的研究过程、主要使用的方法、研究计划表等. 本文主要探究针对蝙蝠 CT 图像的分割与 3D 表面建模的方法. 论文主要由下面几部分组成.

第 1 章 综述. 这章首先简要着对仿生声学及蝙蝠这一背景作简单的介绍, 接着引入本课题的研究背景及目的. 最后, 在本章后半部分还介绍了整个研究过程主要使用的方法、流程以及时间计划表.

第 2 章 原始数据概述. 这一章主要介绍由 CT 显微扫描得到的蝙蝠骨骼切片图像, 这些图像是整个课题研究的起点. 本章主要介绍这些图像的特征, 尤其是对后续研究产生不利影响的缺陷等, 如: 图像模糊、抖动、切片错位等问题–这些问题都是本课题中需要克服和解决的问题.

第 3 章 图像预处理. 这一章主要介绍在对 CT 切片进行分割之前需要进行的一些预处理操作, 例如去除 CT 系统自动生成的系统信息, 去除扫描过程中的无关干扰物对应的图像, 对骨骼图像进行锐化处理等. 本章分为预处理目标、使用的算法、实验及其效果等部分. 本章还会对这个过程使用的算法进行评价和分析.

第 4 章 图像分割. 图像分割是这个课题研究的第一个主要组成部分. 本章首先简要介绍进行分割的目的, 接着叙述如何针对这个特定的课题将分割问题简化, 最后给出分割的结果以及分割集的使用方法.

第 5 章 合成三维模型. 在本章中, 首先给出 VTK 格式的叙述, 接着对使用的工具及软件平台进行叙述, 最后对经过前面数章处理得到的切片图像进行合成, 得到蝙蝠骨骼的三维模型.

第 6 章 三维表面模型建模.3D 表面模型建模是本课题的第二个重要组成部分. 在本章中, 首先给出建模的目的, 接着介绍 L-S Mesh 建模方法. 最后对这一方法进行修正, 并将其运用于蝙蝠骨骼的表面模型建模中.

最后一部分是总结和展望. 在这一部分中, 将对整个课题的研究进行全面的总结, 并提出当中的问题, 给出下一步的研究重点和研究方向.

第 2 章　原始数据

从本章起, 本课题的所有研究课题都是基于对蝙蝠局部骨骼的 CT 图像进行处理和建模, 因此对 CT 扫描所得的原始图像进行分析就显得很有必要了. 本章首先介绍这些原始图像的来源, 扫描设备的局限性等; 接着介绍由于各种原因所导致的图像的缺陷, 说明这些问题可能会对下一步的研究所造成的不良影响; 最后说明本文接下来的部分所需要完成的具体工作, 实施方案及预期效果.

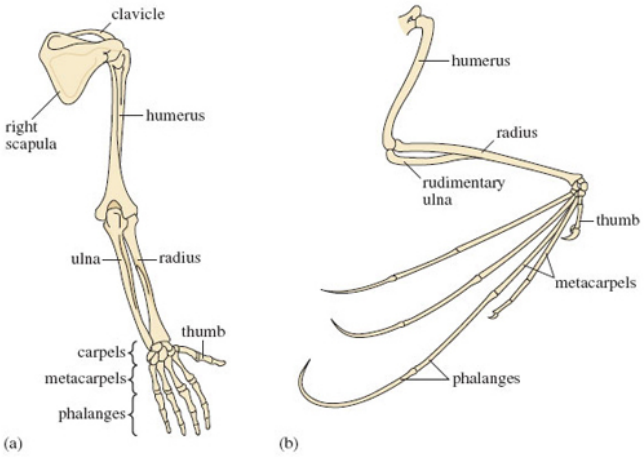


图 2.1 人类手部 (a) 及蝙蝠前肢 (b) 示意图,(b) 为本课题中研究对象所在位置. 图片来源: 英国公开大学课程网站

2.1　原始数据来源

本课题的是实验数据来源于对蝙蝠前肢的计算机断层扫描 (CT) 显微图像. 扫描之前我们已经获得了蝙蝠前肢部分的标本. 为使标本能够被扫描, 我们将其

放入一个透明的塑料容器中, 然后再使用机器对其进行扫描. 扫描的结果是骨骼的切片图像.

理论上讲, 研究人员下一步只要按照顺序将这些图像合成起来即可得到扫描物体的 3D 图像, 但由于 CT 扫描过程中的各种原因与干扰因素, 直接合成的 3D 图像将会是极其的模糊. 因此, 下一节将会介绍 CT 扫描的大体原理, 通过此我们就可以知晓这些图像存在哪些问题, 进而有的放矢地对其进行处理.

2.2 原始数据描述

由于实验设备的限制, 前肢的扫描被分为上下两部分. 上部部分有 1288 张图像, 下部分有 1369 张图像. 在第五章合成三维模型之前, 上下部分可分离处理, 因此之前的操作都是对单张图像的处理. 在合成三维模型是第一步也是分别对上下两部分进行处理, 然后才使用软件将其可并到一个整体中.

对于单张的 PNG 图像, 其宽度和高度均为 1248 像素, 位深度为 8, 通道数为 3. 图 (2.2) 为其中的一张图像.

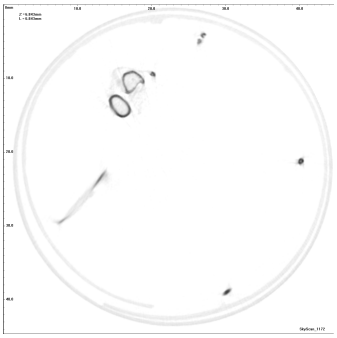
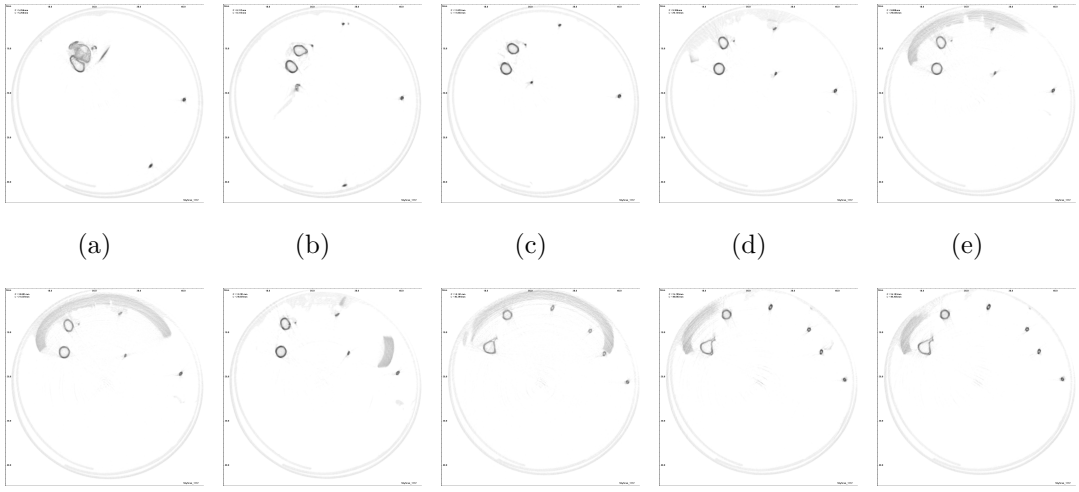


图 2.2 单张图像

由于本研究课题的前半部分都是对单张切片图像进行处理, 为方便对算法及其效果进行描述, 在此我们将选出一些有代表性的图像. 在后文中, 若无特殊的说明, 我们都以下面这些图处理效果来衡量对所有切片图像的处理效果. 这些图像称为 sample 集, 共计 10 张, 如图 (2.3) 所示.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

图 2.3 具有代表性的切片图像 –sample 集

2.3 图像处理需求

在图 (2.3) 中, 我们发现这些图像中有很多的缺陷与问题, 正是这些问题才导致了直接合成三维模型变得不可能, 而本文的前半部分正是试图找到针对与这类切片图像的处理算法, 以便于以后的后续研究工作. 在本节中, 我们将会说明图像处理的需求. 后面的章节将会一一实现这些目标.

一. 原始图像为 PNG 图像, 且为三通道图像. 但由于 CT 扫描的图像性质, 我们希望图像为灰度图, 因为灰度图的处理算法更为容易和有效. 所以我们需要将图像转为灰度图;

二. 每一幅图像中均包含一些 CT 机器的系统信息, 目前我们并不需要这些信息, 所以需要将其去除;

三. 在本章第 (2.1) 节中已说明: 在扫描过程中, 标本是放置在一个塑料容器中的, 而这个塑料容器也出现在了切片图像中, 如图 (2.3) 中的每一个子图的圆形浅灰色区域就是塑料容器. 特别地, 我们发现有一部分该区域的图像已经与标本非常接近或重合, 如图 (2.3(e)) 的左上角 ([8mm-30mm]:[0mm-15mm] [1] 处) 有一个很明显的灰色带, 在后续的处理中要特殊考虑到此种情形, 以使算法具有

[1]表示原图像中长 8mm 至 30mm, 宽 0mm 至 15mm 的矩形区域, 下同

普适性;

四. 由于动物骨腔中存在填充物, 所以在切片图像的骨骼内部存在灰色的填充部分 (如图 (2.3(a)) 的 [13mm-20mm]:[9mm-12mm] 处). 但是, 在后续的三维建模中我们并不需要这些填充物, 需要本课题需要提出去除这部分灰色区域的算法;

五. 从图 (2.3) 中我们可以发现骨骼外围有很多冗余的灰色附加物, 而且骨骼本身的边缘也不是很清晰, 在后面的处理过程也要给出这个问题的解决方案.

上面介绍了原始数据来源, 格式及其存在的问题, 接下来的数章我们就是要解决这些问题. 需要特别提出的是我们并非针对每一个图像处理需求给出特定的算法, 有的算法可能同时可以解决数个问题, 有的问题需要多个步骤才能处理完成, 下文中有介绍.

下一章起即为本文的核心部分. 第三章将介绍对图像进行预处理的操作, 这部分操作都是很简单的去噪和锐化操作, 但其效果会直接影响到后续的算法; 第四章将介绍图像分割, 分割的目的主要是便于后面对单块骨骼的建模; 第五章介绍如何使用软件合成三维模型; 第六章针对前面的三维模型进行优化和简化.

第 3 章　图像预处理

在上一章中我们已说明, 单纯地将断层扫描得到的显微图像进行分割和和合成是不合理的, 因为 CT 切片中有太多的冗余信息和噪声. 因此, 在对其进行分割和合成之前, 我们有必要对这些图像进行预处理. 本章主要介绍这些预处理的流程, 算法, 具体操作操作步骤及其效果.

在前文中已说明, 为方便行文, 我们将以 Sample 集中的切片图像为代表说明和衡量整个算法的效果.

3.1　预处理目标

在2.3节中, 我们知道原始图像存在着很多的问题. 其中本章主要要解决的是:

一. 将图像转为单通道图像, 切片图像实为灰度图, 原图像为三通道, 并不利于后续算法的处理;

二. 消除每个图像边界的系统信心, 这些内容若不消除会干扰后续的分割和合成;

三. 消除塑料容器对应的内容 (详见2.3节);

四. 消除骨腔中的填充物, 后续的分割和合成只关注骨骼外壁和内壁.

五. 突出骨骼的边缘, 这样有利于后面的操作.

下面将分别介绍实现这些目标的方法, 操作及效果.

3.2 图像处理库 -OpenCV

OpenCV (Open Source Computer Vision Library)[3] [4], 是一个跨平台的计算机视觉库. OpenCV 由英特尔公司发起并参与开发，以 BSD 许可证授权发行，可以在商业和研究领域中免费使用. OpenCV 可用于开发实时的图像处理、计算机视觉以及模式识别程序.

OpenCV 库为程序员提供了简单易操作的图像处理接口, 使用 OpenCV 能大大减少变成的代码量, 提供效率. 截止本文最后提交, OpenCV 的最新发行版本是 2014 年 4 月 25 日发布的 OpenCV 2.4.9. 而在本课题中使用的 OpenCV 版本是 2.4.0

本课题的主要开发环境是 Microsoft Visual Studio 2008. 关于如何在 VS2008 中配置 OpenCV, 此处从略, 具体内容可参见 stackoverflow 中的一篇博文[1][5].

3.3 转为灰度图

3.3.1 单通道与三通道

对于一副图像, 它可以为灰度图或者彩色图: 所谓 “灰度图”, 即每个像素只有一个采样颜色的图像, 一般图像显示为最暗的黑色至最亮的白色的灰度[2]; 彩色图的每一个像素颜色更为丰富, 一般分为红 (R) 绿 (G) 蓝 (B) 三个分量.

对于灰度图, 其通道数通常为 1, 彩色图通道数通常为 3–因为灰度图的每一个像素只需一个数值表出即可, 彩色图则需要 3 个.

很显然, 在我们这个课题中, 蝙蝠骨骼的 CT 扫描图像的本质是一个灰度图. 但是在2.2节中我们知道, 原始图像保存的却是三通道图像. 对于这个问题, 三通道的必要性并不大, 所以第一步我们需要将所有的切片图像转为灰度图.

[1]此文章为 OpenCV 2.4.5 版本配置方法, OpenCV 2.4.0 的配置与其无异

[2]灰度图像不同于二值图像, 后者的每个像素只有两个取值, 一般是黑 (0) 和白 (1)

3.3.2 OpenCV 中通道数转化

使用 OpenCV, 我们可以很方便地将原图转为灰度图, 甚至可以在打开图像时同时完成! 具体方法如下:

```
#include <opencv2/opencv.hpp>
using namespace cv;

Mat img;
img = imread( pstrImageName, 0 ); //第一个参数为文件名
```

通过上面这个函数就可以从文件中读取一个图像. 由于 imread 的第二个参数设置为 0, 在打开时图像同时转变图灰度图. 当最后图像操作完成并且保存时, 其通道数即变为 1.

3.4 去除边框信息

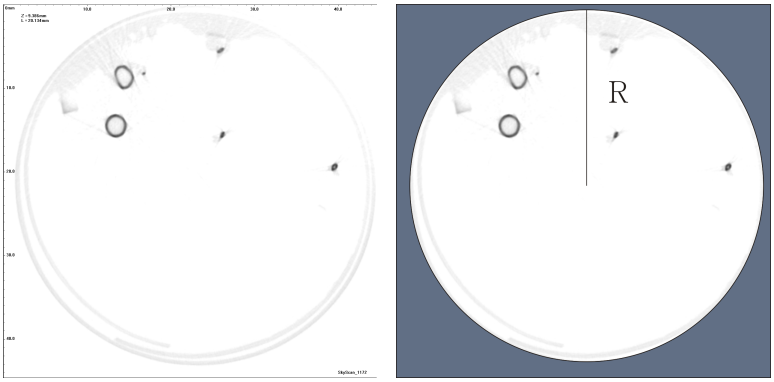
在图 (2.3) 中我们发现, 每一个切片图像的边框都带有一些系统信息 –如图片的标尺, 序号, 机器信息等, 这些信息在合成时并不需要. 本节将介绍如何将这些内容去除.

3.4.1 方法与实现

通过观察, 结合实际情况, 我们发现所有有用的信息 (骨骼对应的图像) 都位于图片中央的一个圆形区域内, 因为扫描时骨骼就是放在一个圆柱状的塑料容器中的. 介于此, 我们可以对于每一张图片 (正方形), 以及中央为原点, 设定一个半径 R, 在这个圆内部的内容得以保留, 在这个圆之外的内容统一置为背景色. 示意图见图 (3.1)

这样一来, 消除边框信息只需要做一个简单的像素因素操作就完成了, 圆形区域内的像素值保持不变, 圆形区域之外的变为背景色 (8 位单通道灰度图白色即为 255). 在实际操作中, 我们把 R 设为 50pixels. 具体代码实现如下:

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
#include <math.h>
using namespace std;
```



(a) (b)

图 3.1 左图为原图; 右图中, 在半径 R 的圆之外的部分 (阴影所示) 可直接设置为背景色 –白色

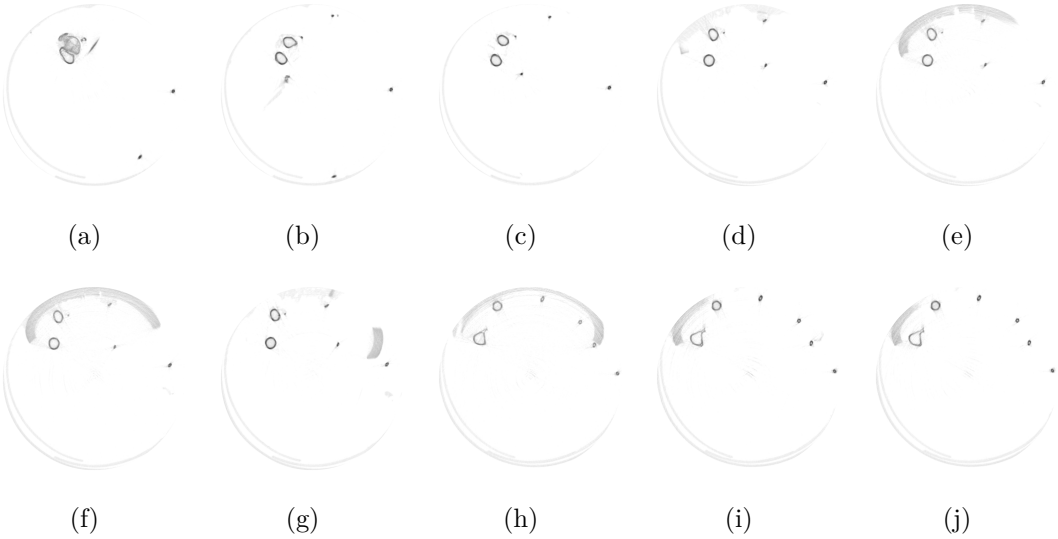
```
using namespace cv;

//消除边界系统信息功能函数
Mat& eliminateSystemInfoInImage(Mat &input){
        int R = hRows - 50; //R=50
        int i,j;
        uchar *p = input.data;
        for ( i = 0; i < nRows; i++ ){
                p = input.ptr<uchar>(i);
                int leftBegin = hCols - sqrt((double)R*R-(i-hRows
                    )*(i-hRows));
                int rightEnd = hCols + sqrt((double)R*R-(i-hRows)
                    *(i-hRows));
                for ( j = 0; j < nCols; j++ ){
                        if(j < leftBegin || j > rightEnd)
                //系统信息域
                                // p[j] 指向(i,j)像素点
                                p[j] = 255;//置成背景色
                }
        }
        return input;
}
```

对于这个方法, 其复杂度为 $O(n^2)$.

3.4.2 执行效果

按照前文所述的, 这里给出 Sample 集的执行效果, 其他切片的执行效果相似. 见图 (3.2)



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

图 3.2 去除边框系统信息: 执行结果

3.5 去除其余噪声

截至上一节, 我们已经去除了一部分多余的信息, 接下来我们将要对切片图像进行一定的操作, 去除绝大部分扫描中产生的噪声等, 具体包括每副图像的圆形边框, 骨腔周围的没用内容以及骨腔内部的填充物.

现在我们分析图 (3.2) 的结果, 不难发现, 整个图像中有用的信息只有骨骼部分. 由于骨骼部分对 X 射线的吸收能力较强 [6], 因而生成的图像灰度值也较大. 对于其余的噪声, 我们可以看到其颜色都比较浅. 基于此, 本文使用了灰度变换这种方法来处理这些噪声. 从下文中将会看到, 这种方法对于这个特定的图像处理问题效果非常好, 而且在去除噪声的同时, 凸出了骨骼的边缘. 这样十分有利于下一步的工作.

3.5.1 直方图与灰度变换

灰度直方图（histogram）是灰度级的函数，它表示图象中具有每种灰度级的象素的个数，反映图象中每种灰度出现的频率。灰度直方图的横坐标是灰度级，纵坐标是该灰度级出现的频率 [3]，是图象的最基本的统计特征.

图 (3.3) 是灰度直方图的一个例子, 它展示了图像 hiparm_06_ud_rec_sam_05.png 的直方图, 其中的横坐标为灰度值 (0-255), 纵坐标为特定灰度值所对应的像素的总个数统计. 由于图片的背景是白色, 所以最后一个 255 的数值应该是数值应该是最大的, 且远大于前面的数值, 为协调图片的比例, 最后一个数值已被略去.

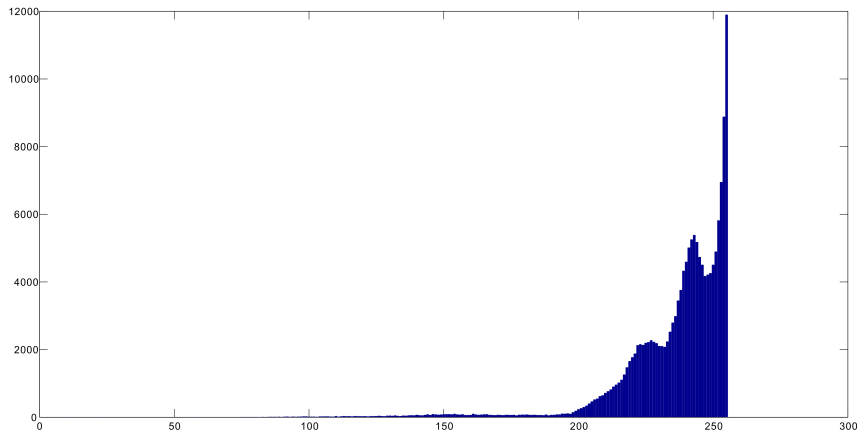


图 3.3 hiparm_06_ud_rec_sam_05.png 对应的灰度直方图. 注意最后一个灰度 255 由于数字太大, 已被略去

灰度变换 [7], 就是找到一个函数 $y = T(x)$, 自变量为原图像的灰度值, 因变量是变换后的灰度值. 将这个函数作用于原图中的每一个元素, 那么每一个元素都会得到一个新的灰度值. 这个过程即为灰度变换.

在本节中, 我们将通过观察和探索直方图建立一个灰度变换函数, 然后利用这个函数对图像进行灰度变换. 需要特别说明的是, 下文没有建立一个通用的函

[3]在本课题中, 由于所有的切片图像都是相同的大小, 为减少计算误差, 下文直接将频数和频率等同起来

数模型, 而是针对每一幅图像计算出其自身的灰度变换函数.

3.5.2 灰度变换模型与参数设定

第一步, 我们假定 T 是一个三次多项式函数[4], 即

$$y = ax^3 + bx^2 + cx + d \tag{3-1}$$

其中 a, b, c, d 为待定参数.

第二步, 计算待定系数. 很明显, $T(0) = 0$, $T(255) = 255$. 实践中, 我们还发现 $T(0.5 * 2555) = 0.5 * 255$, 具体分析见下文.
这样我们就获得了 3 个方程. 见图 (3.3) 中我们不难看出灰度较浅的区域 (自变量较小的区域) 占有的比重并不高. 我们前期统计发现这些区域仅仅占全部区域的 6%. 那么我们可以在统计量为 6% 处所对应的灰度值设为 255*2%(一个很小的量, 以在图像中消除他们).

通过这样的方法, 我们就可以得到 4 个特殊的点, 通过待定系数法很容易地求出变换函数 T.

一点小技巧: 由于自变量数值较大 (大至 255^3), 为减小计算中产生的误差, 我们令,

$$\tilde{x} = \frac{x - 255}{255}$$

因此,

$$x = 255 - 255 \times \tilde{x}$$

在计算过程中, 统一使用 $\tilde{x}$, 完成后在通过第二个式子将其变为 x 即可. 同时因变量也进行类似的操作.

第三步, 将上面计算得到灰度变换函数作用于原图像即可.

[4]使用 Photoshop 等软件可以很容易地手动设定灰度变换函数, 通过实践和观察, 结合实验机器的计算能力, 本文设定灰度变换为三次多项式函数

3.5.3 实现与执行

在上一节中, 我们建立了消除图像噪声的模型. 其中很重要的一步是计算累计频数为 6% 的灰度值. 这一个并不难, 使用 Matlab, 我们很容易就能计算出来, 详细的代码实现见下:

```
function threshold = getThreshold(imgName, proportion)
im = imread(imgName);
hist_im = imhist(im);
a = sum(hist_im)*proportion;
threshold = 1;
b = 0;
while(b < a)
    b = b + hist_im(threshold);
    threshold = threshold + 1;
end
threshold = (255 - threshold) / 255;
```

计算完成后, 我们就有四个特定的点: $T(0)=0, T(1)=1, T(0.5)=0.5, T(threshold)=0.02$, 将其带入式 (3-1) 即可求得 a, b, c, d

还是以前文的 hiparm_06_ud_rec_sam_05.png 为例子, 在原有直方图的基础上, 我们计算其 threshold 值, 不难算出, 为 0.1765.

代入, 求得: $a=-3.3284, b=4.9926, c=-0.6642, d=0$

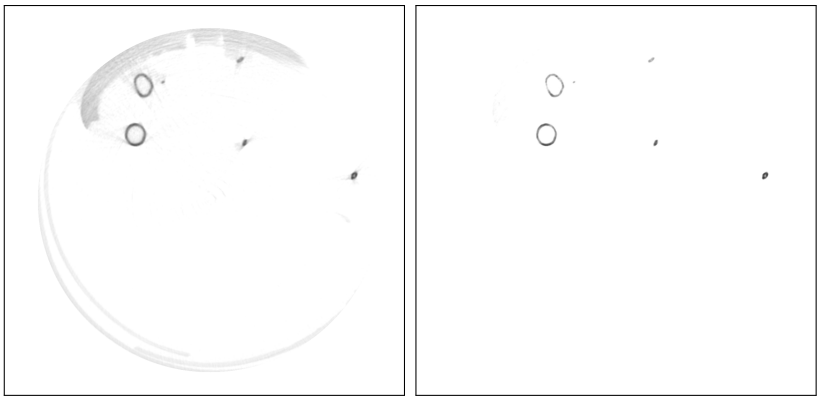
作用于原图, 我们即可得到变换后的图像, 见图 (3.4)

3.5.4 结果分析

现在给出 Sample 集中图像的执行结果, 见图 (3.5)

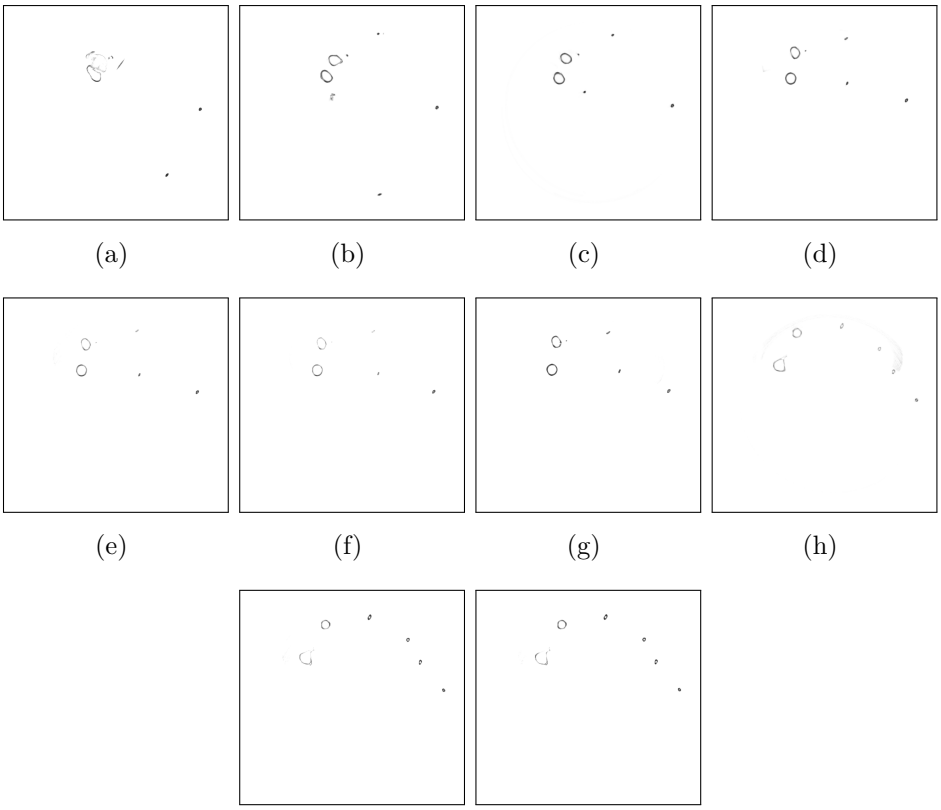
从实验结果看, 切片图像中大部分的信息均已被消除. 究其原因, 我们可知, 由于大部分的多余信息均由于其物理性质的原因导致灰度比较低, 当 $T(treshold)$ 设置为一个很小的值时, 在变换函数的作用下他们就被消除了. 同时由于 a 的值均为负数, 即三次函数的凹凸性在 50% 发生变化, 少于 50% 的内容被弱化, 颜色变得更浅. 大于 50% 的骨骼部分因变量值均大于自变量值, 颜色加深. 这样, 我们通过这个过程, 不仅消除了其余大部分的噪声, 同时使骨骼部分的图像颜色得到了加强.

下一章我们将讨论对蝙蝠骨骼图像进行分割.



(a) (b)

图 3.4 左图为原图; 右图为消除噪声后的效果



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j)

图 3.5 消除其他噪声: 执行结果

第 4 章　图像分割

4.1　图像分割的概念

在图像处理研究过程中, 人们往往只对图像的某些部分感兴趣, 这些部分通常被称为目标或者感兴趣区域 (region of interest, ROI), 图像处理的重要目的就是为了对图像中的目标进行分析, 进而获得目标的客观信息并建立对图像的相关描述. 因此, 在一副图像中, 把目标从背景中分离出来以便于后续处理, 即称为图像分割. 更精确地说, 图像分割是指根据灰度, 颜色, 纹理, 形状等特征吧图像划分成若干互不相交的区域, 使得这些特征在同一区域内呈现出相似性或一致性, 而在不同区域呈现出明显的差异. 借助集合的概念, 对这一定义给出数学描述.[8]

令集合 I 表示一副图像, 对图像 I 的分割可以看作将 I 分成 N 个满足五个条件的非空子集 $\{R_1, R_2, \ldots, R_N\}$:

$$I = \bigcup_{i=1}^{N} R_i$$

$$R_i \bigcap R_j = \emptyset, \forall i, j \wedge i \neq j$$

$$P(R_i) = True, i = 1, 2, \ldots, N$$

$$P(R_i) \bigcup R_j = Flase, i \neq j$$

$$R_i \text{是连通区域}, i = 1, 2, \ldots, N$$

4.2　切片图像分割的意义

在图 (3.5) 中我们可以看到, 每一幅切片在进行预处理之后都是由一个一个的小区域组成的. 事实上, 一个这样的区域就隶属于一块骨骼, 参见图 (2.1(b)).

为此, 如果我们在二维的切片图像中将这样的一个个的区域分割开来, 合成时两个切片之间只关注同一块骨骼的图像. 这样程序就可以自动生成某一块骨骼的三维图像, 而非整个扫描标本所对应的三维图像.

4.3 问题的转化

图像分割一般的方法包括分水岭方法, 金字塔分割, 均值漂移, 水平集等方法. 这些方法适用于前景与背景区分并不是很明显, 分割集合较多的情况. 虽然这些图像分割方法研究已经比较成熟, 效果也很好, 甚至有很多已经在 OpenCV 库函数中被实现. 但是, 这些方法执行效率低, 有时候还存在过度分割等问题, 在此课题中并不适用.

经过观察不难发现, 在进行上一章的图像预处理之后, 图片的前景和背景已经区分得非常明显了. 如果我们**将这个图片按照一定的灰度阈值将其变为二值图像**, 那个整个分割过程就已经基本上完成一大半了.

将图像转化为二值图像后, 接下来仅需要求出二值图像的强连通分支即可.

4.4 实现与执行

按照上一节所述, 先将原图变为二值图像. 为最大限度地保留原图信息, 我们将阈值设为 200. 因此有下面的函数[1]:

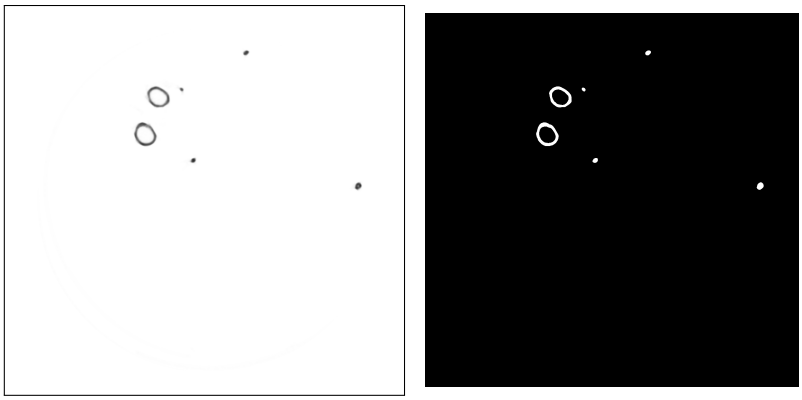
$$f(x) = \begin{cases} 1 & \text{if } x < 200 \\ 0 & \text{if } x \geq 200 \end{cases} \tag{4-1}$$

这个过程程序实现非常容易, 此处从略. 图 (4.1) 是其中一个例子.

接下来求强连通分支.

本文使用**两遍扫描法**. 所谓两遍扫描法, 就是通过扫描两遍图像, 将图像中存在的所有连通区域找出并标记. 具体思路是: 第一遍扫描时赋予每个像素位置一个标记 label, 扫描过程中同一个连通区域内的像素集合中可能会被赋予一个或多个不同 label, 因此需要将这些属于同一个连通区域但具有不同值的

[1]假定 1 为前景, 0 为背景



(a) (b)

图 4.1 左图为 hiparm_06_ud_rec_sam_03.png; 右图为其对应的二值图像, 白色部分为前景

label 合并, 也就是记录它们之间的相等关系; 第二遍扫描就是将具有相等关系的 label 所标记的像素归为一个连通区域并赋予一个相同的 label（通常这个 label 是那些相同标记中的最小值).

算法步骤:

Step I. 第一次扫描:

访问当前像素 Img(x,y)，如果 Img(x,y) == 1:

case 1. 如果 Img(x,y) 的领域中像素值都为 0, 则赋予 Img(x,y) 一个新的 label: label++, Img(x,y) = label

case 2. 如果 Img(x,y) 的领域中有像素值大于 1 的像素 Neighbors[2]:

1）将 Neighbors 中的最小值赋予给 Img(x,y): Img(x,y) = minNeighbors

2）记录 Neighbors 中各个值（label）之间的相等关系，即这些值（label）同属同一个连通区域: $labelSet[i]$= { $label_m$, .., $label_n$}，$labelSet[i]$ 中的所有 label 都属于同一个连通区域

Step II. 第二次扫描:

访问当前像素 Img(x,y), 如果 Img(x,y) > 1: 找到与 label = Img(x,y) 同属相等关系的一个最小 label 值，赋予给 Img(x,y).

完成扫描后, 图像中具有相同 label 值的像素就组成了同一个连通区域.

[2]四邻域或八邻域均可, 本文使用四邻域

使用 C++ 可以很快地实现这个算法, 代码见附录A.

4.5 分割结果的使用

执行上面一节的过程后, 我们就可以得到每一个切片的分割数据. 在上一章的合成中, 如果我们想合成蝙蝠骨骼的某一块骨头, 只需要识别相邻切片同一块骨头的分割区域, “屏蔽” 其他部分即可完成这个操作. 当然, 直接在第3章之后的基础上完成这个合成也是可以的, 只是不能马上得到某一块特定的骨头罢了.

为此, 我们需要保存上一节算法中的 label 集合及其对应的像素点, 由于切片很多, 需要的存储空间很大, 这是本算法的其中一个缺点.

关于识别上下两个切片中哪个区域同属一块骨头, 方法很简单: 只需要对每一个切片计算出各个分割集坐标的平均位置, 然后对比上下两个切片, 位置相近了就一定是同一块骨骼, 因为在这个实际问题中, 切片图像是不会有突变的.

下一章我们将介绍如果合成三维模型, 这个过程主要是通过一个软件平台实现的.

第 5 章　合成三维模型

在上一章当中我们介绍了切片图像的分割. 在完成二维图像的预处理和分割操作之后, 我们就可以顺利地读取这一系列的二维切片渲染成三维骨骼. 课题要求将数据表述为 VTK 格式. 本章首先介绍三维可视化工具箱, 接着介绍 VTK 文件格式, 最后使用这个工具箱完成合成操作.

5.1　VTK 工具箱概述

VTK [9](Visualization ToolKit) 是一个开放源码, 自由获取的软件系统, 全世界的数以千计的研究人员和开发人员用它来进行 3D 计算机图形, 图像处理, 可视化. VTK 包含一个 c++ 类库, 众多的翻译接口层, 包括 Tcl/Tk, Java, Python.

Visualization Toolkit 是一个用于可视化应用程序构造与运行的支撑环境, 它是在**三维函数库 OpenGL**的基础上采用**面向对象**的设计方法发展起来的, 它将我们在可视化开发过程中会经常遇到的细节屏蔽起来, 并将一些常用的算法封装起来[1].

VTK 给从事可视化应用程序开发工作的研究人员提供直接的技术支持的一个强大的可视化开发工具, 它以用户使用的方便性和灵活性为主要原则, 具有如下的特点：

1) 具有强大的三维图形功能: Visualization Toolkit 既支持基于体素 Voxel-

[1]比如 Visualization Toolkit 中包含 vtkMarchingCubes 类, 这个类就提供了表面重构中经常用到的 Marching Cubes 算法, 这样在对三维规则点阵数据进行表面重建时就不必再重复编写 MarchingCubes 算法的代码, 而直接使用 Visualization Toolkit 中已经提供的 vtkMarchingCubes 类

basedrendering 的体绘制 Volume Rendering 又保留了传统的面绘制, 从而在极大的改善可视化效果的同时又可以充分利用现有的图形库和图形硬件
2) VTK 的体系结构使其具有非常好的流 streaming 和高速缓存 caching 的能力, 在处理大量的数据时不必考虑内存资源的限制
3) VTK 能够更好的支持基于网络的工具比如 Java 和 VRML. 随着 Web 和 Internet 技术的发展 VTK 有着很好的发展前景
4) 能够支持多种着色如 OpenGL 等
5) VTK 具有设备无关性使其代码具有良好的可移植性
6) VTK 中定义了许多宏，这些宏极大的简化了编程工作并且加强了一致的对象行为
7) VTK 具有更丰富的数据类型，支持对多种数据类型进行处理
8) 既可以工作于 Windows 操作系统又可以工作于 Unix 操作系统极大的方便了用户

VTK 项目的官网 `http://www.vtk.org/` 提供了更多相关的介绍.

`http://www.vtk.org/doc/nightly/html/index.html` 中有 VTK 相关的文档资料.

5.2 VTK 体数据类型

5.2.1 结构化数据和非结构化数据

可视化数据 (如图像像素点) 可以分为规则 (Regular) 和不规则 (Irregular) 或者说结构化 (Structured) 和非结构化 (Unstructured). 规则结构数据点之间有固定的关联关系, 可以通过这些关联确定每个点的坐标, 不规则结构数据之间没有固定的关联关系.

对于规则结构的数据, 存储时不必存储所有的数据点, 只需存储起始点、相邻两点之间的间隔以及点的总数就可以保存完整的数据信息. 对于不规则结构的数据, 虽然不可以像规则结构的数据那样存储, 但它也有自身的优势: 即在数据变化频繁的区域可以密集表示, 而数据变化不频繁的区域则稀疏表示. 规则结构的数据可以在存储及计算时占优势, 不规则结构的数据虽然存储和计算时不能像规则结构的那样高效, 但它在数据表达方面相对而言则更加自由, 更加细致、灵活的表现数据.

考虑到本课题后面还需要对骨骼表面模型进行重建, 此处直接使用**结构化数据 (Structured Data)**.

5.2.2 vtkDataSet 的组织结构

vtkDataSet 的组织结构由拓扑结构 (Topology) 和几何结构 (Geometry) 两部分组成. 拓扑结构描述了物体的构成形式, 几何结构描述了物体的空间位置关系. 也就是说, 点数据 (Point Data) 所定义的一系列坐标点构成了 vtkDataSet(数据集) 的几何结构; 点数据的连接[2]就形成了数据集的拓扑结构. 比如, 要想在屏幕上显示一个三角形, 首先我们必须定义三角形三个点的坐标 (即 Point Data, 记三个点为 P_1, P_2 和 P_3), 然后将这三个点按照一定的顺序连接起来 (P_1-P_2-P_3, 或者是 P_3-P_2-P_1 的顺序), 这三个点定义了数据集的几何结构, 它们的连接就构成了数据集的拓扑结构. 亦即, **点数据 (Point Data)**定义数据集的几何结构, **单元数据 (Cell Data)**定义数据集的拓扑结构, 要形成完整的数据集, 必须有几何和拓扑两种结构.

5.2.3 vtkDataSet 可视化数据类型

vtkDataSet 有六种结构类型, 分别是 Structured Points(笛卡儿网格), Rectilinear Grid(矩形网格), Structured Grid(结构网格), Unstructured Points(散乱点), Polygonal Data(规整网格) 和 Unstructured Grid(非结构网格). [10] 一书中有对这几种类型作详细介绍.

5.3 合成骨骼表面模型

前面数节已经对 VTK 及其数据类型作了介绍, 下面直接给出本课题中生成 VTK 的程序. VTK 库使用简洁方面, 效率很高. 代码原来及主要功能在程序的注释中也做了说明.

```
#include "vtkDICOMImageReader.h"
#include "vtkImageShrink3D.h"
#include "vtkDataObject.h"
```

[2]点的连接先形成单元数据 (Cell Data), 由单元数据再形成拓扑

```
#include "vtkMarchingCubes.h"
#include "vtkDecimatePro.h"
#include "vtkSmoothPolyDataFilter.h"
#include "vtkPolyDataNormals.h"
#include "vtkStripper.h"
#include "vtkPolyDataMapper.h"
#include "vtkCamera.h"
#include "vtkActor.h"
#include "vtkProperty.h"
#include "vtkRenderer.h"
#include "vtkRenderWindow.h"
#include "vtkWin32RenderWindowInteractor.h"
#include "vtkVolume16Reader.h"

int
skeletonSynthesis(){

vtkDICOMImageReader *reader = vtkDICOMImageReader::New();
        reader->SetDataByteOrderToLittleEndian();
        reader->SetDirectoryName("D:/Working/BTProject/down");

vtkVolume16Reader *v16 = vtkVolume16Reader::New();
   v16->SetDataDimensions(1248, 1248);
   v16->SetDataByteOrderToLittleEndian();
   v16->SetFilePrefix("D:/Working/BTProject/down/
       hiparm_06_ud_rec_");//工作路
       径
   v16->SetFilePattern("%s%.4d.png");
   v16->SetImageRange(0,50);

vtkImageShrink3D *shrink = vtkImageShrink3D::New();
        shrink->SetShrinkFactors(4,4,1);
        shrink->AveragingOn();
        shrink->SetInput((vtkDataObject *)v16->GetOutput());

vtkMarchingCubes *skinExtractor = vtkMarchingCubes::New();
        skinExtractor->SetValue(0,300);
```

```
        skinExtractor->SetInputConnection(shrink->GetOutputPort()
            );

vtkDecimatePro *deci = vtkDecimatePro::New();
        deci->SetTargetReduction(0.3);
        deci->SetInputConnection(skinExtractor->GetOutputPort());

vtkSmoothPolyDataFilter *smooth = vtkSmoothPolyDataFilter::New();
        smooth->SetInputConnection(deci->GetOutputPort());
        smooth->SetNumberOfIterations(200);

vtkPolyDataNormals *skinNormals = vtkPolyDataNormals::New();
        skinNormals->SetInputConnection(smooth->GetOutputPort());
        skinNormals->SetFeatureAngle(60.0);

vtkStripper *stripper = vtkStripper::New(); //将三角形连接起来
        stripper->SetInputConnection(skinNormals->GetOutputPort()
            );

vtkPolyDataMapper *skinMapper = vtkPolyDataMapper::New();
        skinMapper->SetInput(stripper->GetOutput());
        skinMapper->ScalarVisibilityOff();

vtkCamera *aCamera = vtkCamera::New();
        aCamera->SetViewUp(0,0,-1);
        aCamera->SetPosition(0,1,0);
        aCamera->SetFocalPoint(0,0,0);
        aCamera->ComputeViewPlaneNormal();

vtkActor *coneActor = vtkActor::New();
        coneActor->SetMapper(skinMapper);
        coneActor->GetProperty()->SetDiffuse(1);
        coneActor->GetProperty()->SetSpecular(0.6);


//下面是显示相关代码
vtkRenderer *renderer = vtkRenderer::New();
        renderer->AddActor(coneActor);//添加对象coneActor
```

```
        //renderer->AddActor2D(textActor)添加;//对象textActor
        renderer->SetBackground(0,0,0);
        renderer->SetActiveCamera(aCamera);//添加照相机
        renderer->ResetCamera();

vtkRenderWindow *renWin = vtkRenderWindow::New();//设置绘图窗口
        renWin->AddRenderer(renderer);//装载绘图类

vtkWin32RenderWindowInteractor *iren =
    vtkWin32RenderWindowInteractor::New();
        //iren->SetRenderWindow(renWin)装载绘图窗口;//
        iren->Initialize();
        iren->Start();

return 0;
}
```

5.4 合成结果

上一节给出合成 vtk 数据的程序. 但由于上面的文件是 binary 二进制文件, 要观察其结果还需要自己编写程序, 且不能调整光照, 比例等. 使用 vtkPLYWriter 类可以很方便地将 VTK 格式转化为 PLY 格式 [11]. 而 PLY 文件格式就能用 MeshLab 打开并观察了.

5.4.1 PLY 格式文件

PLY 文件格式是斯坦福大学开发的一套三维 mesh 模型数据格式, 图形学领域内很多著名的模型数据, 比如 Stanford 的三维扫描数据库, Geogia Tech 的大型几何模型库，北卡（UNC）的电厂模型等最初的模型都是基于这个格式的.

PLY 多边形文件格式的开发目标是建立一套针对多边形模型的, 结构简单但是能够满足大多数图形应用需要的模型格式, 而且它允许以 ASCII 码格式或二进制形式存储文件. PLY 的开发者希望, 这样一套既简单又灵活的文件格式, 能够帮助开发人员避免重复开发文件格式的问题.

PLY 作为一种多边形模型数据格式, 不同于三维引擎中常用的场景图文件

格式和脚本文件, 每个 PLY 文件只用于描述一个多边形模型**对象 (Object)**, 该模型对象可以通过诸如顶点、面等数据进行描述, 每一类这样的数据被称作一种**元素 (Element)**. 相比于现代的三维引擎中所用到的各种复杂格式, PLY 实在是种简单的不能再简单的文件格式, 但是如果仔细研究就会发现, 就像设计者所说的, 这对于绝大多数的图形应用来说已经是足够用了.

PLY 的文件结构同样很简单: 文件头加上元素数据列表. 其中文件头中以行为单位描述文件类型、格式与版本、元素类型、元素的属性等, 然后就根据在文件头中所列出元素类型的顺序及其属性, 依次记录各个元素的属性数据.

当然, 既然我们已经获得了 VTK 文件, 那么 PLY 格式文件就可以直接调用 vtkPLYWriter 类来获得了.

5.4.2 立体网状处理软件 -MeshLab

MeshLab[3]是一个用 QT 开发的、开源的、功能比较强大的可扩展的网格处理系统, 致力于辅助清理、适配、编辑和显示非结构化的 3D 三角形网格, 尤其适合于处理 3D 扫描得到的网格. 它的核心是使用 VCG 库[4], MeshLab 和 VCGlib 都是意大利国立大学可视计算实验室发布的. 图 (5.1) 是 MeshLab 的操作界面 (图片来源: `http://meshlab.sourceforge.net/`).

5.4.3 合成效果

在第4章第4.5节中我们讨论了如何使用分割而来的数据合成单一的一块骨头. 图 (5.2) 给出其中一块骨头的合成效果. 其他的骨骼合成结果类似, 在下一章中的处理过程也类似, 为保持全文的一贯性, 下一章中的三维表面模型重构使用图 (5.2) 中的关节部位.

下一章将介绍本课题中的最后一个步骤 –三维表面模型重建.

[3]源代码 `http://sourceforge.net/p/meshlab`

[4]VCG 是一个 C++ 的 3D 网格处理库: `http://sourceforge.net/projects/vcg/`

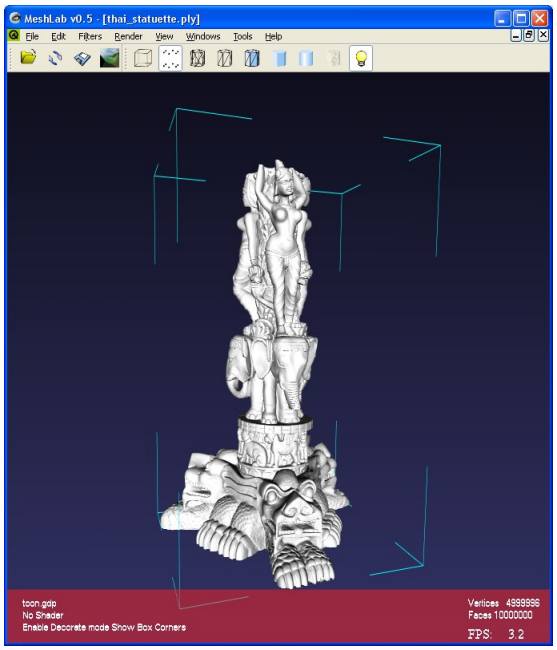


图 5.1 MeshLab 的操作界面

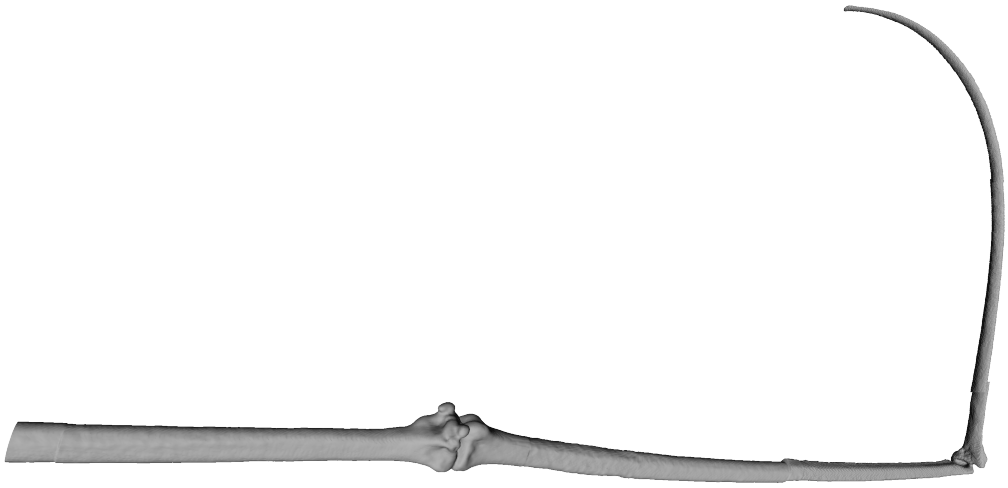


图 5.2 三维表面合成结果之一

第 6 章　三维表面模型重建

6.1　建模目的

截至上一章, 我们已经将骨骼的三维表面模型建立出来了. 但是, 在上一章中建立的骨骼表面模型虽说精度较高, 但由于其顶点和平面很多, 非常不利于今后研究, 因为其需要大量的存储空间和消耗大量的运算资源. 比如说, 图 (5.2) 展示的只是其中一块骨骼, 它是由 22 万多个顶点, 40 多万个平面组成, PLY 文件大小达 8MB 多. 至于整个标本, 其 PLY 文件合计达到 226MB.

鉴于此, 本章提出一种三维表面模型压缩的算法, 试图在保持骨骼表面形状不变的情况下, 尽可能地压缩顶点数目和平面数目, 以减少模型的大小.

6.2　LS-Mesh 方法

2004 年 Olga Sorkine、Daniel Cohen-Or 等人在 [12] 中提出了一个网格建模方法, 称之为 Least-Squares Meshes (简称 LS-Mesh 方法). 这种方法思想简单, 操作方便, 却十分的有效, 非常适合于本课题中的三维表面模型数据压缩. 本文的附录B提供了 [12] 英文原文和中文译文, 对于这个方法的正确性证明请参阅该处. 本节主要介绍 LS-Mesh 的主要思想和方法.

对于一个图 (Graphic) $G = (V, E)$, 我们令 $V = \{1, 2, \ldots, N\}$ 为其顶点序列, v_i 是顶点 i 的位置 (未知). 则, 满足下面式子的顶点我们就称之为光滑的 (平衡的).

$$v_i - \frac{1}{d_i} \sum_{j:(i,j)\in E} v_j = 0 \tag{6-1}$$

其中 d_i 是顶点 i 的度 (degree). 如果式 (6-1) 被满足, 则点 i 位于其邻接顶点的几何中心.

若所有点都是平衡的话, 则有:

$$Lx = 0, Ly = 0, Lz = 0$$

其中,x, y, z 都是 $n \times 1$ 的向量, 为 n 个点的 x, y, z 轴坐标. L 为 $n \times n$ 的矩阵:

$$L_{ij} = \begin{cases} 1 & i = j \\ -\frac{1}{d_i} & (i,j) \in E \\ 0 & \text{其它的} \end{cases}$$

L 就是网格的拉普拉斯算子, 其秩为 $n - k$ (k 是图 G 的连通分支数).

讨论至今, 我们仍然只是关注了网格的顶点的关联关系 (用于计算 L 矩阵). 显然, 这样是不合理的: 我们还需要向模型提供一定的顶点位置信息. LS-Mesh 的创新之处就在于, 我们只需要提供很少量的顶点的位置信息 (这些顶点即为控制顶点 control points), 然后就可以计算出在最小二乘意义下最平滑的曲线 – 这正是 LS-Mesh 的主要思想. 对于其正确性和合理性的更多说明, 可参阅附录B.

下面介绍如何在前文所述的模型基础上添加控制顶点 (control points).

假定给出 m 个控制顶点, 记为

$$v_s = (x_s, y_s, z_s), s \in C.$$

其中 $C = \{s_1, s_2, \ldots, s_m\}$ 是控制顶点.

方程的系数矩阵就可变为 $((n + m) \times n)$:

$$Ax = \mathbf{b} \tag{6-2}$$

其中

$$A = \begin{pmatrix} L \\ F \end{pmatrix}, F_{ij} = \begin{cases} 1 & j = s_i \in C \\ 0 & \text{其它的} \end{cases}$$

$$b_k = \begin{cases} 0 & k \leq n \\ x_{s_{k-n}} & n < k \leq n+m \end{cases}$$

图 (6.1) 给出了 A 矩阵的一个例子,

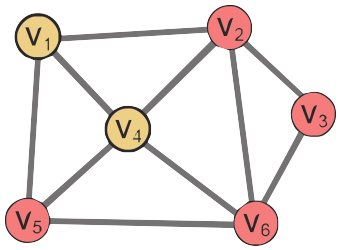


图 6.1 一个有向图, V_1 和 V_4 为控制顶点

其对应的系数矩阵应为:

$$\begin{pmatrix} 1 & -\frac{1}{3} & 0 & -\frac{1}{3} & -\frac{1}{3} & 0 \\ -\frac{1}{4} & 1 & -\frac{1}{4} & -\frac{1}{4} & 0 & -\frac{1}{4} \\ 0 & -\frac{1}{2} & 1 & 0 & 0 & -\frac{1}{2} \\ -\frac{1}{4} & -\frac{1}{4} & 0 & 1 & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{3} & 0 & 0 & -\frac{1}{3} & 1 & -\frac{1}{3} \\ 0 & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

得到一个线性系统后, 我们求出式 (6-2) 在最小二乘意义下的解, 即我们要找到一个向量 x, 使得下值最小化

$$\|Ax - b\|^2 = \|Lx\|^2 + \sum_{s \in C} |x_s - v_x^{(x)}|^2 \tag{6-3}$$

同理得到 y 和 z, 3 个方程组的解就是得到的顶点的坐标.

6.3 Control Points 选取

行文至此, 整个算法的关键就剩下如何选取控制点 (Control Points) 了. 前文已说过, 控制点为三维模型重建提供了仅有的少量位置信息. 为此, 控制点应

当因可能地选取一些" 关键" 的点处, 比如网格的边缘, 拐角, 凸角或凹角等.

控制点的选取有多种方法, 最简单的一种方法就是随机选取. 确定控制点所占的比例 (如 5%) 后, 随机在所有顶点中选出这个比例的点作为控制点. 这种方法操作最简单, 算法执行速度也是最快的. 但缺点也十分明显: 由于是随机选取, 所以算法每次执行的结果具有随机性, 而且很有可能选出的顶点不能落在模型的" 关键点" 处. 介于此, 我们应当找到一个" 策略" 选取顶点. 具体方法如下所述:

从某一个随机点开始, 每次新增一个新的控制顶点. 具体地, 以目前已有的控制点作为一个 LS Mesh 的控制顶点, 然后接触一个新的 LS Mesh, 比较新得到的 LS Mesh 与原有的 Mesh 的位置差异, 将位置差异最大的那个点作为新的控制顶点.

这种方法每次都需要解一个线性系统, 算法耗时比较长, 效果比较好, 基本上能找到" 关键" 的控制点.

但是, 再深入考虑, 不难发现, 上述控制点的选取方法运算量太大, 不利于算法的推广. 因此, 我们应该找到一种更为实际的, 运算量更为适中的折中方案. 具体地,

算法每执行 K(如 30) 步进行一次上述方法的运算, 得到一个控制点. 其余的 $(K-1)$ 步生成 $(K-1)$ 个点, 方法是:

从最新得到的控制点开始对网格进行宽度优先搜索遍历, 找到所有从控制点开始 $error$[1]递减的所有路径. 然后在这些路径之外的点中取出 $error$ 最大的一个点作为控制点. 重复上述过程 $(K-1)$ 次得到 $(K-1)$ 个顶点.

6.4 实验过程与结果

基于运算能力和算法复杂度考虑, 本实验选取图 (5.2) 骨骼中间最复杂的一部分作为研究对象, 如图 (6.2) 所示.

仅仅是图 (6.2) 中的片段, 其就包含了 54174 个顶点, 这显然需要很大的存储空间. 下面我们使用 LS Meshes 方法对其进行重建. 实验中我们需要选出 1500 个顶点作为控制顶点.

[1] $error$ 等于 LSMesh 与原 Mesh 的差距

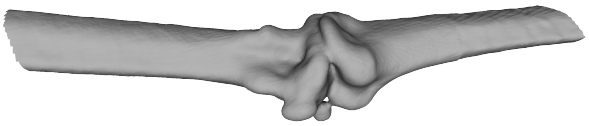


图 6.2 实验片段

第一步, 根据 PLY 文件提供的数据得到其关联图各个顶点的度数.
第二步, 选出一个顶点[2]作为控制点, 然后计算新的 LS-Meshes.[3].
第三步, 计算 LS-Mesh 和原有 Mesh 的差距. 以差距最大的点作为新的控制点.
第四步, 以新的控制点作为起点对图进行 BFS, 找到所有 error 递减的路径, 并排除. 然后取出剩余点中 error 最大的点作为新控制点. 第五步, 重复第四步 29 次.
第六步, 重复第二至五步 50 次.
最后, 得到 1500 个控制点, 以此计算 LS-Mesh.

最后的计算结果, 如图 (6.3) 所示.

6.5 分析与讨论

上面的整个计算过程中, 所耗费时间最多的就是解 LS-Mesh 的线性最小二乘系统. 而这当中最主要的时间都花在了就矩阵进行 LU 分解上面. 在上一节的实验中, 我们需要进行 50 次 LU 分解, 平均每次耗费 3.21s.

但是, 我们也应该看到, 对于一个固定的模型, 其控制点的计算是一次性的, 也就是说, 我们执行的 50 次 LU 分解也是一次性的. 在今后的实际课题运用中,

[2]第一个出发点可以是随机一个点, 如 v_1
[3]此处使用了 LUG@USTC 提供的 sagemath 计算平台进行计算

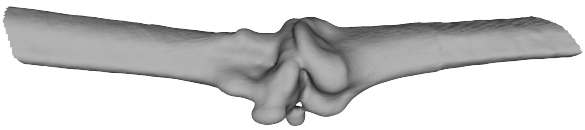


图 6.3 图 (6.2) 对应的 LS-Mesh, 控制点数目: 1500

控制点是固定的, 模型的关联矩阵是固定的, 所以 LU 分解也是可以一次性 (预先完成) 完成. 如果线性系统的右侧向量发生变化 (如变为某些物理量), 我们只需做一个回代和前代即可知道其他点对应的值.

关于新模型的存储, 我们需要存储控制点的坐标, 各个顶点的连通关系, 其余的在运用前即时计算即可. 在控制点数目只有原 Mesh 2.7% 的情况下, 我们发现模型的形状基本不发生变化. 由此可见这个建模方法的合理性.

经过这样一个过程, 整个模型的存储量大大减少了, 我们在存储和计算量之前找到了一个平衡 –使用少量的计算获得了很大的存储优化.

对于为何 LS-Mesh 能仅仅使用如何少的控制点的位置信息就可以重构整个模型这个问题, 我们可以观察图 (6.4), 在其某些较为规则的部分, 原 Mesh 的点分布还是很密集的 (图上的红色点所示, 其中有 13174 个点, 占整个模型的四分之一), 事实上, 这些点的位置是可以由很少量的控制点通过某些规则计算出来的.

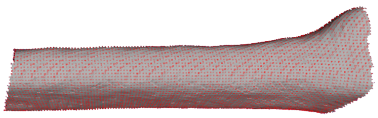


图 6.4 图 (6.2) 某个部分的顶点示意图

第 7 章　总结与展望

本文从蝙蝠肢部骨骼的 CT 显微切片出发, 给出了图像分析, 图像预处理, 分割, 合成和优化三维模型整个研究过程. 在图像分析一章中, 我们对整个课题的原始数据进行了评估和分析; 紧接着我们有使用了不同方法对图像进行预处理, 以使其达到合成的要求. 之后我们针对实际情况, 给出了一个很简单的分割方法, 然后对切片进行了合成, 生成了 VTK 文件 (进而转化为 PLY 文件). 得到三维模型之后, 本文有尝试使用 LS Meshes 方法对得到的三维模型进行优化和简化, 从效果上看尚可.

由于生物骨骼本身的构造很复杂, 所以所有的分割和优化等基本上都是有人为介入的部分. 例如本课题中不管对图像进行如何的处理, 在最后合成后还是有一些小碎片出现在三维模型中, 这些碎片其实就是前期处理不妥造成的, 我们只有对其进行人工删除. 接下来的研究中, 我们有必要针对医学图像处理这一特定的领域给出更为优越的图像处理方法, 而非使用普遍的常规的算法.

另外, LS Meshes 在本文中也只是一个尝试, 不管怎么优化, 其运算量仍然是相当的大. 找到一种更好的模型重建方法仍然是待完成的工作.

致　　谢

四年的本科生活在这个季节即将划上一个句号, 而于我的人生却只是一个逗号, 我将面对又一次征程的开始. 四年的求学生涯在师长、亲友的大力支持下, 走得辛苦却也收获满囊, 在论文即将付梓之际, 思绪万千, 心情久久不能平静.

四年来, 山东大学严谨的学风、安静的校园环境让我燃起了对计算机、软件学习的激情. 通过四年的学习, 我对计算机的理解已经发生巨大的变化, 我透过对计算机体系、计算机理论与编程语言等方面的学习体会到了计算机思考 (Thinking in computer science) 的严谨与乐趣. 因此我非常感激山东大学软件学院的老师在这四年中对我的关心和指导, 是他们改变了我以往思考计算机和思考日常问题的方式和方法.

最后, 本毕业设计是在我的导师潘荣江教授的亲切关怀和悉心指导下完成的. 他严谨的科学态度, 精益求精的工作作风, 诲人不倦的高尚师德, 严以律己、宽以待人的崇高风范, 平易近人的人格魅力深深地感染和激励着我. 从课题的选择到项目的最终完成, 潘老师都始终给予我细心的指导和不懈的支持, 在此谨向潘老师致以诚挚的谢意和崇高的敬意.

同时, 我还要:

感谢软件学院的老师对我的教育培养, 他们细心指导我的学习与生活;

感谢 LUG@USTC 为我的实验提供了服务器计算集群, 有了他们的帮助我的毕业设计才能得以顺利完成;

感谢给我提供参考文献的学者们, 谢谢他们给我提供了大量的文献, 使我得到了最新的技术与方法;

感谢大学四年与我朝夕相处的同学和朋友, 感谢他们在这四年对我的帮助和支持;

感谢我的爸爸妈妈, 感谢他们为我所付出的一切. 养育之恩, 无以回报, 你们永远健康快乐是我最大的心愿.

四年学习即将结束, 我又将踏上新的学习征程, 未来总是充满着机遇与挑战! 祝福明天!

参考文献

[1] Danilo Emilio De Rossi and Michael Pieroni. Grand challenges in bionics. *Frontiers in Bioengineering and Biotechnology*, 1:3, 2013.

[2] Rolf Müller, Hongwang Lu, and John R Buck. Sound-diffracting flap in the ear of a bat generates spatial information. *Physical review letters*, 100(10): 108701, 2008.

[3] Opencv wikipedia. `http:zh.wikipedia.org/wiki/OpenCV`, . Accessed: 2014-03-20.

[4] Opencv.org. `http://opencv.org/`, . Accessed: 2014-03-20.

[5] Installation of opencv on visual studio 2008. `http://stackoverflow.com/questions/16574959/installation-of-opencv-2-4-5-on-visual-studio-2008`. Accessed: 2014-04-05.

[6] 百度百科 -x 射线. `http://baike.baidu.com/view/45735.htm`. Accessed: 2014-04-18.

[7] KYJ Zhang and PETER Main. Histogram matching as a new density modification technique for phase refinement and extension of protein molecules. *Acta Crystallographica Section A: Foundations of Crystallography*, 46(1): 41–46, 1990.

[8] 阮秋琦. 数字图像处理学. 电子工业出版社, 2007.

[9] I Kitware. Vtk user’s guide version 5, kitware. *Inc., Sep*, 2006.

[10] 周振环, 郑小中, 赵明. 三维图像编程实验. 电子工业出版社, 书号: 7-121-14881-1/TP.1516, 北京, 2011.

[11] Paul Bourke. Ply-polygon file format, 2009.

[12] Olga Sorkine and Daniel Cohen-Or. Least-squares meshes. In *Shape Modeling Applications, 2004. Proceedings*, pages 191–199. IEEE, 2004.

[13] 严洪范. Ct——计算机断层扫描. 自然杂志, 12:007, 1984.

[14] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia. Meshlab: an open-source 3d mesh processing system. *Ercim news*, 73:45–46, 2008.

[15] James A Sethian. Fast marching methods. *SIAM review*, 41(2):199–235, 1999.

[16] Kenton McHenry and Peter Bajcsy. An overview of 3d data content, file formats and viewers. *National Center for Supercomputing Applications*, 1205, 2008.

附录 A　两遍扫描法代码

```
#include <iostream>
#include <string>
#include <list>
#include <vector>
#include <map>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace std;
using namespace cv;

void icvprCcaByTwoPass(const Mat& binImg,Mat& lableImg){
  // 第一次扫描
  lableImg.release();
  binImg.convertTo(lableImg, CV_32SC1);

  int label = 1;
  vector<int> labelSet;
  labelSet.push_back(0);   // 背景: 0
  labelSet.push_back(1);   // 前景: 1

  int rows = binImg.rows - 1;
  int cols = binImg.cols - 1;
  for (int i = 1; i < rows; i++){
        int* data_preRow = lableImg.ptr<int>(i-1);
        int* data_curRow = lableImg.ptr<int>(i);
        for (int j = 1; j < cols; j++){
          if (data_curRow[j] == 1){
```

```
            vector<int> neighborLabels;
            neighborLabels.reserve(2);
            int leftPixel = data_curRow[j-1];
            int upPixel = data_preRow[j];
            if ( leftPixel > 1) //左
                    neighborLabels.push_back(leftPixel);
            if (upPixel > 1) //上
                    neighborLabels.push_back(upPixel);

            if (neighborLabels.empty()){
                    labelSet.push_back(++label);  // 新标号
                    data_curRow[j] = label;
                    labelSet[label] = label;
        }
        else {
                    sort(neighborLabels.begin(),
                        neighborLabels.end());
                    int smallestLabel = neighborLabels[0];
                    data_curRow[j] = smallestLabel;

                    // 合并同值
                    for (size_t k = 1; k < neighborLabels.
                        size(); k++){
                            int tempLabel = neighborLabels[k
                                ];
                            int& oldSmallestLabel = labelSet[
                                tempLabel];
                            if (oldSmallestLabel >
                                smallestLabel){
                                    labelSet[oldSmallestLabel
                                        ] = smallestLabel;
                                    oldSmallestLabel =
                                        smallestLabel;
                            }
                            else if (oldSmallestLabel <
                                smallestLabel){
                                    labelSet[smallestLabel] =
                                         oldSmallestLabel;
```

```
                                    }
                                }
                            }
                        }
                    }
                }

                for (size_t i = 2; i < labelSet.size(); i++){
                    int curLabel = labelSet[i];
                    int preLabel = labelSet[curLabel];
                    while (preLabel != curLabel){
                        curLabel = preLabel;
                        preLabel = labelSet[preLabel];
                    }
                    labelSet[i] = curLabel ;
                }


                // 第二次扫描
                for (int i = 0; i < rows; i++){
                    int* data = _lableImg.ptr<int>(i);
                    for (int j = 0; j < cols; j++){
                        int& pixelLabel = data[j];
                        pixelLabel = labelSet[pixelLabel];
                    }
                }
            }
```

附录 B　外文参考文献及译文

本文的第6章使用了 LS Meshes 这种建模方法. 这种建模方法简单而富有成效. 此处给出其英文原文及其中文译文.

Least-squares Meshes

Olga Sorkine
Tel Aviv University
sorkine@tau.ac.il

Daniel Cohen-Or
Tel Aviv University
dcor@tau.ac.il

Abstract

In this paper we introduce Least-squares Meshes: meshes with a prescribed connectivity that approximate a set of control points in a least-squares sense. The given mesh consists of a planar graph with arbitrary connectivity and a sparse set of control points with geometry. The geometry of the mesh is reconstructed by solving a sparse linear system. The linear system not only defines a surface that approximates the given control points, but it also distributes the vertices over the surface in a fair way. That is, each vertex lies as close as possible to the center of gravity of its immediate neighbors. The Least-squares Meshes (LS-meshes) are a visually smooth and fair approximation of the given control points. We show that the connectivity of the mesh contains geometric information that affects the shape of the reconstructed surface. Finally, we discuss the applicability of LS-meshes to approximation of given surfaces, smooth completion and mesh editing.

1 Introduction

This paper introduces Least-squares Meshes: meshes that are constructed from a given connectivity and approximate a set of control points in a least-squares sense. Given a planar graph with arbitrary connectivity and a sparse set of control points with geometry, we reconstruct the geometry of the rest of the mesh vertices by solving a sparse linear system. The linear system not only defines a surface that approximates the given control points, but it also distributes the vertices over the surface in a fair way. That is, each vertex lies as close as possible to the center of gravity of its immediate neighbors. The LS-meshes are a visually smooth and fair approximation of the given control points.

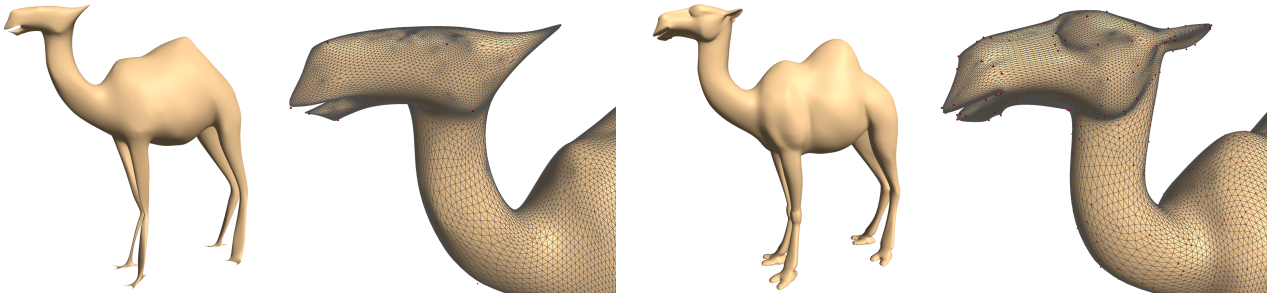
This paper also shows that by carefully selecting the control points, an LS-mesh can effectively approximate a given mesh. Figure 1 displays LS-meshes with the connectivity and control points of the camel model. Figure 2(a) shows a horse model consisting of 20K vertices. In 2(b), the same connectivity is used to approximate a subset of 1000 vertices of the model in 2(a). A close-up view of the LS-mesh is shown in 2(d) to demonstrate the fairing effect of the LS-approximation.

Common scattered data approximation techniques [7] receive an unstructured data set of points as input and fit a continuous surface that approximates (or interpolates) the points, while satisfying some desired conditions, such as smoothness properties. For visualization and further processing purposes, the continuous surface is then often sampled and triangulated. When implicit functions are fitted to approximate a set of points [3, 15, 17], they define a level set of a trivariate function from which the surface needs to be extracted. In contrast, an LS-mesh directly fits a prescribed mesh over a surface that approximates the points.

Our work is close to the convex interpolation method of Floater [6], where a given mesh is laid over a plane. As we shall describe in the following section, our method and Floater's method both distribute the mesh vertices fairly, while satisfying constraints given as control points. However, Floater's control points are *hard* constraints, while our control points are *soft* and satisfied in the least-squares sense. Floater's method reconstructs the geometry of the mesh in the plane, while our method reconstructs an arbitrary surface is 3D. In particular, LS-meshes can be constructed from arbitrary connectivity graphs, and generate shapes with genus higher than zero and surfaces which contain boundaries.

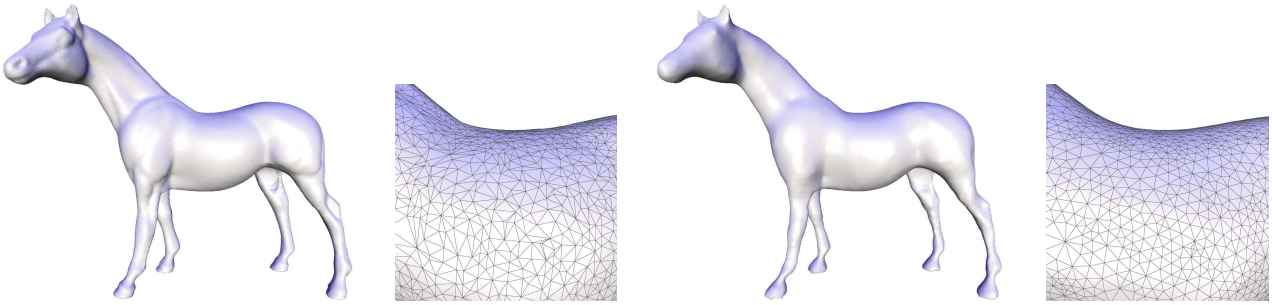
Another related area of work is the variational subdivision schemes [11, 12], which can be used to solve the scattered data interpolation problem. Kobbelt [12] applies Gauss-Seidel iterations to minimize the thin plate energy of the surface [14]. These iterations act as a smoothing filter on the mesh. Interleaving the Gauss-Seidel iteration steps with mesh subdivision steps generates a smooth surface from an initial sparse mesh.

The rest of the paper is organized as follows. The next section reviews the mathematical background needed to construct an LS-mesh. In Section 3 we discuss the properties of LS-meshes. Different strategies to obtain LS-meshes that approximate a given shape are shown in Section 4. Section 5 addresses the algorithmic issues of solving the linear



(a) (b) (c) (d)

Figure 1. LS-mesh: a mesh constructed from a given connectivity graph and a sparse set of control points with geometry. In this example the connectivity is taken from the camel mesh. In (a) the LS-mesh is constructed with 100 control points and in (c) with 2000. The connectivity graph contains 39074 vertices (without any geometric information). (b) and (d) show close-ups on the head; the control points are marked by small dots.



(a) (b) (c) (d)

Figure 2. (a) The original horse model, 19851 vertices; (b) close-up on the original connectivity; (c) LS-mesh of the horse model with 1K control vertices; (d) close-up on the LS-mesh connectivity.

least-squares system. We discuss the results and applications in Section 6 and conclude in Section 7.

2 Overview

Let $G = (V, E)$ be the given mesh graph, where $V = \{1, 2, ..., n\}$ is the set of vertex indices and E is the set of edges. We denote by $\mathbf{v}_i$ the (unknown) location of vertex i in space. The following equation defines a fairness and smoothness condition for vertex $\mathbf{v}_i$ (similar to [6]):

$$\mathbf{v}_i - \frac{1}{d_i} \sum_{j:(i,j)\in E} \mathbf{v}_j = 0, \tag{1}$$

where d_i is the valence of vertex i. If the above equation is satisfied, the vertex i lies in the center of gravity of its immediate neighbors.

Tutte [18] has shown that if we assume the points $\mathbf{v}_i$ to reside in the 2D plane, the above system defines a valid embedding of a planar graph onto a plane, provided that the boundary vertices of the graph are set to lie on the boundary of a convex polygon (and thus the boundary vertices and their corresponding equations are eliminated from the system). Floater [6] extended this result for systems that require the vertices to lie in any convex combination of their neighbors. Here, we would like to solve this system for 3D meshes, thus $\mathbf{v}_i$'s are assumed to be in $\mathbb{R}^3$. The linear system can be written in matrix form:

$$L\mathbf{x} = 0, \quad L\mathbf{y} = 0, \quad L\mathbf{z} = 0,$$

where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are the n×1 vectors containing the x, y and z coordinates of the n vertices and L is the following n×n matrix:

$$L_{ij} = \begin{cases} 1 & i = j \\ -\frac{1}{d_i} & (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

The matrix L is known as the *Laplacian* of the mesh [5, 10, 16]. The rank of L is $n - k$ where k is the

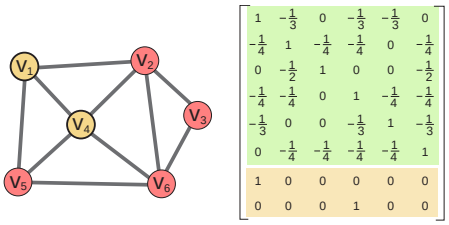


Figure 3. A small example of a graph and its corresponding matrix. The dark vertices are set as control points.

number of connected components in the graph G. Therefore, assuming that our mesh graph is connected, the rank of L is $n-1$, and there is a one-dimensional subspace of solutions for the system spanned by the vector $(1,1,...,1)^T$.

Without any geometric information given, the solution of the system is not interesting. Providing the 3D location for some m *control vertices* allows to get a non-trivial and unique solution. We add the equations of the control vertices:

$$\mathbf{v}_s = (x_s, y_s, z_s), \quad s \in C, \tag{2}$$

where $C = \{s_1, s_2, ..., s_m\}$ is the set of indices of the control vertices. Our system then becomes rectangular ((n+m)×n):

$$A\mathbf{x} = \mathbf{b},$$

where

$$A = \begin{pmatrix} L \\ F \end{pmatrix}, \quad F_{ij} = \begin{cases} 1 & j = s_i \in C \\ 0 & otherwise \end{cases}$$

$$b_k = \begin{cases} 0 & k \le n \\ x_{s_{k-n}} & n < k \le n+m \end{cases}$$

See Figure 3 for an example of a mesh with control vertices and the corresponding matrix. Adding at least one row for a control vertex to our system makes the A matrix full-rank. The system is solved in least-squares sense, i.e. we find $\mathbf{x}$ that minimizes

$$\|A\mathbf{x} - \mathbf{b}\|^2 = \|L\mathbf{x}\|^2 + \sum_{s \in C} |x_s - \mathbf{v}_s^{(x)}|^2. \tag{3}$$

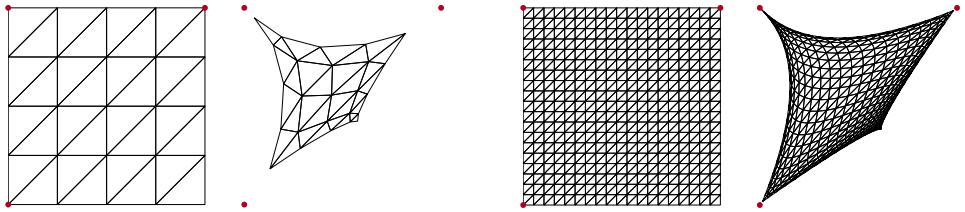
The unique analytical solution is $\mathbf{x} = \left(A^T A\right)^{-1} A^T \mathbf{b}$ since A has full rank. Note that element (i,j) of the matrix $A^T A$ does not vanish only if the graph distance between i and j is at most two. This implies that $A^T A$ is sparse (although not as sparse as A that only accounts for first-order neighborhoods).

Note that the control vertices play a conceptually similar role to the boundary vertices in Tutte's graph embedding solution. They constrain the system by augmenting the connectivity with some geometric information and allow to obtain an interesting solution. However, the major difference between Tutte's and Floater's boundary conditions and our control vertices is the approach to solving the system: while they force the constrained vertices to lie in the exact prescribed location, thus eliminating them from the system, we solve the system in least-squares sense, and the constraints are thus approximated, rather than interpolated. We keep the smoothness and fairness equations for the control vertices, and the resulting solution tends to respect these conditions at the constrained vertices, as well as at the free ones.

To obtain a better understanding and intuition for the claims above, take a look at Figure 4, which shows a simple 2D example. A 5×5 triangulated grid is shown in (a), where three of its vertices are used as control points. The geometric position of the rest of the grid points is meaningless and is only used to illustrate the given connectivity. An LS-mesh constructed from this connectivity with the three control points is shown in the second column. Note that the LS-mesh does not interpolate the three control points, and the vertices are not placed exactly at the average position of their neighbors. The error comes from the fact that there is no solution that can fully satisfy all the 25+3 constraints. When there are more elements in the mesh (see the 20×20 triangulated grid in (b)) the mesh is more flexible and the error is better distributed among the constraints. In particular, the control points are better satisfied.

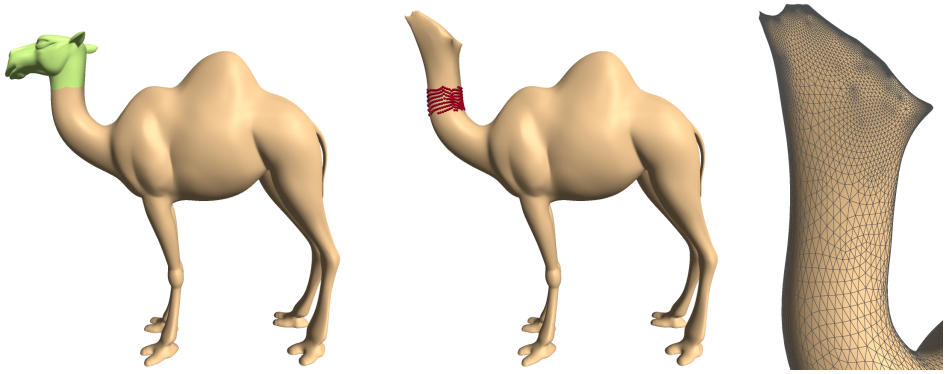
3 Connectivity Meshes

A 3D mesh consists of its connectivity information and the geometry, i.e., the 3D coordinates of the mesh vertices. We will refer to a mesh that has no geometry information as a connectivity mesh. A connectivity mesh has no shape. It can be generated from an arbitrary mesh by removing its geometry information. In our work we deal with what we can call an augmented connectivity mesh, where only a subset of the mesh vertices contains geometric information. Our linear least-squares system reconstructs the geometry of the mesh vertices while approximating the known geometry of the subset, and positions each vertex in the mean geometry of its immediate neighbors. Since the reconstruction system also accounts for the given connectivity of the mesh, it yields a shape which is close to the notion of connectivity shapes [9]. In their work, Isenburg et al. showed that a pure connectivity mesh has some natural shape, assuming that all the edges of the mesh are of equal length. They employ an iterative optimization process which minimizes an energy functional that inflates the mesh towards a smooth shape where the edges are close to uniform length. The optimization process is non-linear and requires to interleave some regularization steps so that the reconstructed shape



(a) 5×5 grid (b) 20×20 grid

Figure 4. LS-meshes of triangulated grids. The first and third columns visualize the connectivity of the given graphs; the geometric position of the grid vertices is meaningless. Three control points at the corners of the grid are used to generate the LS-meshes, their positions are marked by small dots. Note that since the LS-mesh satisfies the constraints in least-squares sense, some error is present: neither the control points are interpolated nor the fairness condition is precisely satisfied.



(a) (b) (c)

Figure 5. Reconstructing the geometry of the camel's head using the original connectivity. We removed the geometry from the head (marked in (a)). (b) The control points around the "hole" are marked by small spheres. (c) Close-up on the reconstructed geometry. Note that the connectivity of the head contains some information that induces non-trivial shape, without using a single control point.

bears some resemblance to the original mesh. However, the key contribution of their work is that it shows that a pure connectivity mesh contains some non-trivial geometric information.

In light of the above, an LS-mesh can be regarded as a non-pure connectivity mesh, where only some of its vertices contain geometric information. If the given connectivity mesh is meant to reconstruct a given shape, these geometry vertices act as control points assisting the reconstruction process to yield a better approximated shape. While the connectivity shapes strive to satisfy the uniform edge-length condition, in an LS-mesh the vertices satisfy flatness and fairness conditions in a least-squares sense. It is interesting to note that the smoothness term used as regularization in the optimization process of connectivity shapes [9] is in fact the Laplacian of the mesh.

When a small set of control points is used, the LS-mesh is reminiscent of the connectivity shapes in the sense that LS-mesh as well draws some non-trivial geometric information from the connectivity alone. This is demonstrated in Figure 7, where the legs on the left are reconstructed from a very sparse set of control points, and and in Figure 5, where the head of the camel model is reconstructed from the connectivity graph, using only the ring of vertices around the neck as control points.

4 Selecting the control points

LS-meshes can approximate a given mesh. A set of control points needs to be chosen in order to bring the surface of the LS-mesh close to the original mesh and minimize the geometric error. Intuitively, the control points should be

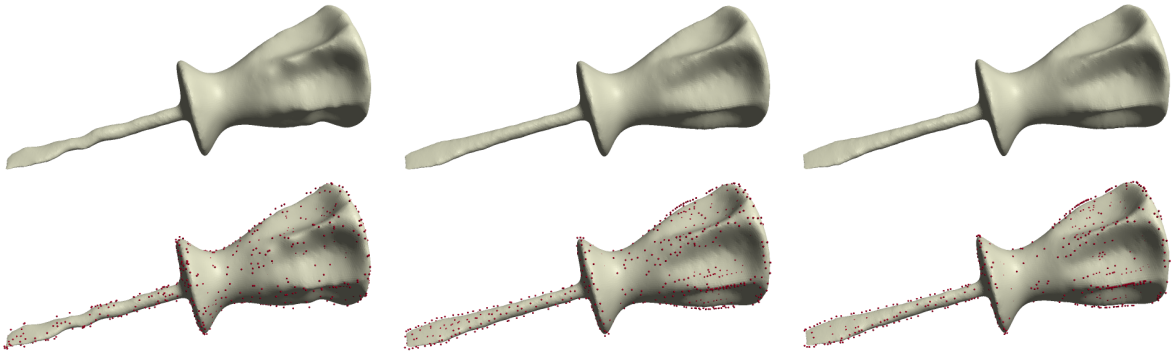


Figure 6. Different approaches to selecting the control points, applied to the screwdriver mesh (27152 vertices, 1000 control points). The bottom row shows the locations of the control points by small dots. In the left column, random selection was used. The middle column displays the greedy approach, which places one control point at a time in the vertex that attained the maximal error. In the right column the combined approach was used, where in each of 31 steps we found 31 local error maxima and placed the control points there, and then recomputed the LS-mesh to obtain the error estimation for the next selection step. One can see that random selection is inefficient since it does not "predict" places which will have large reconstruction error. On the other hand, both greedy selection and the combined approach work quite well and concentrate the control points in strategic regions, such as the edges of the screwdriver, where the reconstruction error is likely to be larger otherwise.

places in "strategic" locations on the surface, such as feature points, the tips of extruding parts and places where geometric detail is present. We have tested several strategies to select the control points:

- Random selection. This is a fast method, however if the original mesh is highly irregular and high-frequency details are present, random sampling might miss them out, which results in an inefficient approximation (see Figure 6(a)).
- One-by-one greedy selection. This method chooses one control point in each step by computing the LS-mesh induced by the current set of control points and placing the new control point at the vertex whose location in the LS-mesh has maximal error compared to its location in the original mesh. This method is slower since it requires solving the least-squares system in each step, but it is successful at identifying the "important" control points (such as points on features and details) and decreasing the approximation error.
- Combined local maxima method. With this strategy, we compute the LS-mesh every K steps and mark the vertex with maximal error as a control point, like in the greedy method. In the $K-1$ steps in between, we select control points by computing local maxima of the error. That is, we traverse the mesh in breadth-first order, starting from the last selected control point (that attains the global maximum of the error) and mark the vertices around it as "forbidden", as long as the error decreases. Then, we find the vertex with the maximal error among the vertices that were not marked. The process repeats $K-1$ times, and then we compute the new LS-mesh. This approach is faster than the greedy method since less solves are required, and it imitates the greedy method by fairly distributing the control points.

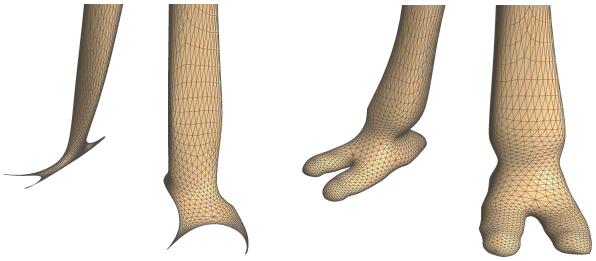
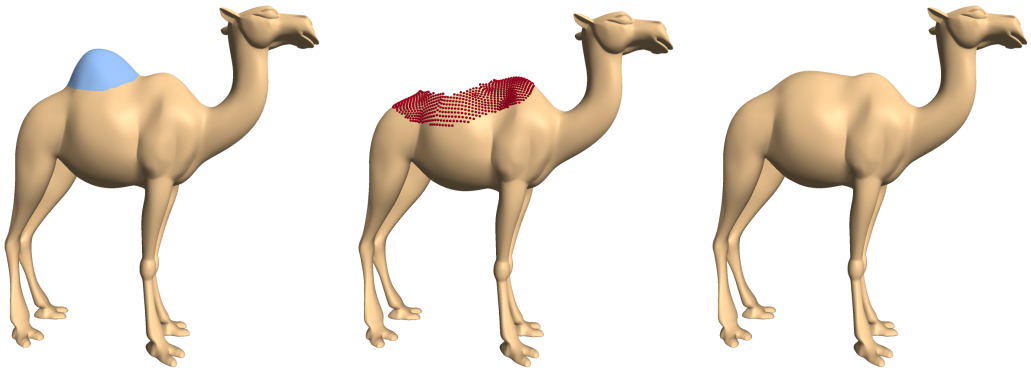


Figure 7. Close-up on the legs of the camel LS-mesh computed using 100 control points (left) and 2000 control points (right).

An example comparing the three approaches in shown in Figure 6. While the greedy approach seems to achieve the best distribution of the control points and hence the smallest geometric error, the combined approach is more practical in making the tradeoff between the approximation error and computation time.



(a) (b) (c)

Figure 8. Reconstructing the geometry of a hole using LS-meshes. (a) shows the original camel model, and the region of the hole is marked. We "removed" the geometry from the vertices of the hump and reconstructed it using the control points (marked by small spheres in (b)) and the connectivity of the hump. The reconstructed model is shown in (c).

Model	# vertices	Factor	Solve	Total
Eight	2,718	0.085	0.004	0.097
Horse	19,851	0.900	0.032	0.996
Screwdriver	27,152	1.646	0.068	1.850
Camel	39,074	2.096	0.073	2.315

Table 1. Running times (sec.) of solving the linear least-squares systems for the different models. *Factor* **denotes the time spent on the factorization of the normal equations.** *Solve* **is the time to solve for one mesh function ($\mathbf{x}$, $\mathbf{y}$ or $\mathbf{z}$). The last column shows the total time to compute the LS-mesh.**

5 Solving the system

LS-meshes require to solve a sparse linear least-squares system to reconstruct the geometry, i.e. to minimize $\|A\mathbf{x} - \mathbf{b}\|$. We use a direct method for solving the normal equations $A^T A\mathbf{x} = A^T\mathbf{b}$. A factorization of the coefficient matrix $A^T A = R^T R$ is found, where R is an upper triangular matrix. Then $\mathbf{x}$ (and $\mathbf{y}, \mathbf{z}$) is found by solving two triangular linear systems $R^T R\mathbf{x} = A^T\mathbf{b}$, that is $R^T\tilde{\mathbf{x}} = A^T\mathbf{b}$ and $R\mathbf{x} = \tilde{\mathbf{x}}$.

Table 1 records the solution times for the models used in our experiments on a 2.4 GHz Pentium 4 computer. The table shows the time to decompose the coefficient matrix of the normal equations into its triangular factors, the solution time for one mesh function ($\mathbf{x}$, $\mathbf{y}$ or $\mathbf{z}$ vectors) and the total solution time. The direct method is quite fast for moderately large meshes. Most of the time is spent on computing the factorization, while the time of the solving is negligible.

6 Discussion

As discussed above, LS-meshes can approximate shapes. As the number of control points gets smaller, the LS-mesh gets closer and closer to being a pure connectivity shape. Figure 11 shows a series of LS-meshes approximating different models with increasing number of control points. When the amount of control points is really small, the LS-mesh is distorted and bears almost no similarities to the original shape. However, as the number of control points increases, the reconstruction quickly gets closer to the original shape. By giving higher weight to the control points constraints, the LS-mesh can become closer to being interpolatory. The weights are incorporated by modifying equation (2) to the following:

$$w_s\mathbf{v}_s = w_s(x_s, y_s, z_s).$$

Thus, the energy minimized by the LS system has now the form

$$\|A\mathbf{x} - \mathbf{b}\|^2 = \|L\mathbf{x}\|^2 + \sum_{s\in C} w_s^2 |x_s - \mathbf{v}_s^{(x)}|^2.$$

As one can see in the 2D example in Figure 9, when the weight of the control points increases, the LS-mesh better approximates them at the expense of the fairness constraints.

LS-meshes can be potentially utilized for filling holes in surfaces. It is possible to adopt a framework similar to the one proposed by Lévy [13]. Lévy conformally embeds the mesh in the 2D plane, triangulates the hole in the parameter domain, and then reconstructs the geometry of the newly

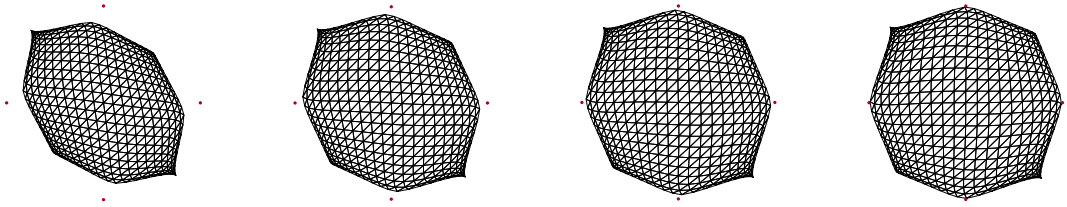


Figure 9. LS-meshes constructed with varying the weight of the control points constraints. The location of the control points is marked by small circles. As the weight increases, the LS-mesh becomes closer to interpolatory, at the expense of compromising the fairness conditions.

introduced vertices by minimizing a discrete curvature criterion. By solving our linear system, where the known vertices around the hole serve as control points, the geometry of the unknown vertices is reconstructed. See Figure 8, where the vertices around the hump were taken as control points, and the hump was reconstructed by solving the system on the reduced mesh (which included the control points and the connectivity of the hump). We do not need a parameterization of the mesh but just a triangulation of the hole. The resulting shape depends on the number of vertices used in the triangulation and the quality of the meshing. For instance, when taking the head of the camel and "removing" the geometry from it, and then reconstructing it by solving our system, we arrive at the non-trivial shape shown in Figure 5.

Another interesting area of application is shape modeling and editing. By manipulating the control points, a new LS-mesh is immediately implied. However, unlike in subdivision methods [19], here the connectivity is readily given and better fits to the subject geometry. This suggests that LS-meshes can be effective for editing existing shapes. See for example the horse in Figure 10, where we moved the control points of its head, resulting in an edited version of the same mesh. The reconstruction of the LS-mesh is very fast, assuming that the factorization of the normal equations matrix of the LS system is pre-computed, which needs to be done only once per LS-mesh. Then, to obtain the edited LS-mesh we only need to solve the system by back-substitution, which is very efficient, as discussed in Section 5 (in particular, refer to the timings in the *Solve* column of Table 1).

It is quite tempting to investigate the potential of LS-meshes for geometry compression and progressive transmission of 3D meshes. Previous work on progressive mesh compression employed construction of hierarchy for both the geometry and connectivity of the mesh [1, 8, 20]. Recent methods have concentrated on progressive representation of the geometry only [4, 10]. In that sense, LS-meshes are somewhat related to the Spectral Compression method [10], since in both cases, the connectivity of mesh

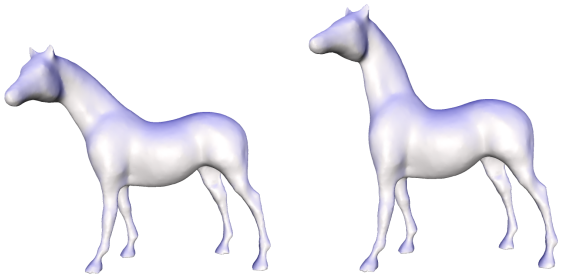


Figure 10. An example of editing the LS-mesh. On the left the original LS-mesh is shown; on the right the resulting LS-mesh after moving the control points on the head.

is assumed to be decoded before the geometry is decoded. Roughly speaking, the spectral method succeeds to reconstruct large models with a visually pleasing appearance using a few thousands of spectral coefficients. In terms of file size, this is equivalent to a few thousands of control points (see Figure 2(c) for the horse model approximated with 1000 control points). A fair comparison would require to agree on a visual metric and to draw distortions curves. While such a comparison is beyond the scope of this paper, we would like to emphasize that spectral methods tend to be optimal [2]. However, a progressive transmission of a triangular mesh based on LS-meshes can be advantageous in terms of its computational cost. The spectral method requires computing the eigenvectors of the mesh Laplacian matrix, which for large meshes is an extremely costly computation. In contrast, the solution of linear least-squares system is fairly simple and direct.

7 Conclusion

LS-meshes are conceptually simple and easy to implement. They provide means to reconstruct smooth meshes from a given connectivity mesh and a (possibly small) num-

ber of control points. The geometry of the mesh is reconstructed by solving a sparse linear system. LS-meshes can deal with high-genus shapes and surfaces which contain boundaries.

It is important to note that the uniform coefficients in the smoothness condition (Eq. 1) depend solely on the connectivity of the mesh. Other forms of discretization can be used if the mesh geometry is known. For example, Kobbelt [12] locally fits quadratic polynomials to the mesh to approximate second order derivatives. Then, the smoothness condition is expressed by non-uniform weights that vary over the mesh depending on the surface geometry and the characteristics of the meshing. In our ongoing research we are looking into various discretizations schemes of the smoothness condition, such as averaging of higher-order neighborhoods.

We feel that this work opens up a lot of new and interesting research directions, which we hope others will join us in exploring. These include:

- Understanding the behavior of the LS solution and the connection to subdivision surfaces. We refer the reader again to Figure 5, where the shape of the camel's head is reconstructed solely from the connectivity. Applying a subdivision scheme, starting from some coarse triangulation of the "chopped" neck, would reconstruct only a smooth cap. The machinery that recovers geometric information from the connectivity needs further exploration.

- Expanding the domain of LS-meshes to shapes with sharp features. Currently, LS-meshes handle only smooth surfaces. We would like to incorporate sharp edges, preferably in similar global LS framework.

- Analyzing the smoothness and the approximation properties of LS-meshes.

- Exploring the potential of LS-meshes for progressive transmission of meshes along the lines discussed above.

Acknowledgements

We would like to thank Sivan Toledo for insightful discussions and Christian Rössl for proofreading. This work was supported in part by grants from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), by the Israeli Ministry of Science, by the German Israel Foundation (GIF) and by the EU research project 'Multiresolution in Geometric Modelling (MINGLE)' under grant HPRN-CT-1999-00117. The screwdriver mesh is courtesy of Cyberware.

References

[1] P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of ACM SIGGRAPH 2001*, pages 198–205, 2001.

[2] M. Ben-Chen and C. Gotsman. On the optimality of spectral compression of meshes. Preprint, available online from *http://www.cs.technion.ac.il/∼gotsman/publications.html*, 2003.

[3] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[4] P. H. Chou and T. H. Meng. Vertex data compression through vector quantization. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):373–382, 2002.

[5] M. Fiedler. Algebraic connectivity of graphs. *Czech. Math. Journal*, 23:298–305, 1973.

[6] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[7] R. Franke and G. M. Nielson. Scattered data interpolation and applications: A tutorial and survey. In H. Hagen and D. Roller, editors, *Geometric Modelling, Methods and Applications*, pages 131–160. Springer Verlag, 1991.

[8] H. Hoppe. Progressive meshes. In *Proceedings of ACM SIGGRAPH 96*, pages 99–108, August 1996.

[9] M. Isenburg, S. Gumhold, and C. Gotsman. Connectivity shapes. In *Proceedings of IEEE Visualization 2001*, pages 135–142, 2001.

[10] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *Proceedings of ACM SIGGRAPH 2000*, pages 279–286, July 2000.

[11] L. Kobbelt. A variational approach to subdivision. *Computer Aided Geometric Design*, 13:743–761, 1996.

[12] L. Kobbelt. Discrete fairing and variational subdivision for freeform surface design. *The Visual Computer*, 16(3-4):142–158, 2000.

[13] B. Lévy. Dual domain extrapolation. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):364–369, 2003.

[14] H. P. Moreton and C. H. Séquin. Functional optimization for fair surface design. In *Proceedings of ACM SIGGRAPH 92*, pages 167–176. ACM Press, 1992.

[15] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003)*, 22(3):463–470, 2003.

[16] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of ACM SIGGRAPH 95*, pages 351–358, 1995.

[17] G. Turk and J. F. O'Brien. Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, Oct. 2002.

[18] W. T. Tutte. How to draw a graph. *Proc. London Mathematical Society*, 13:743–768, 1963.

[19] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann Publishers Inc., 2001.

[20] J. C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *Proceedings of IEEE Visualization '96*, pages 327–334., 1996.

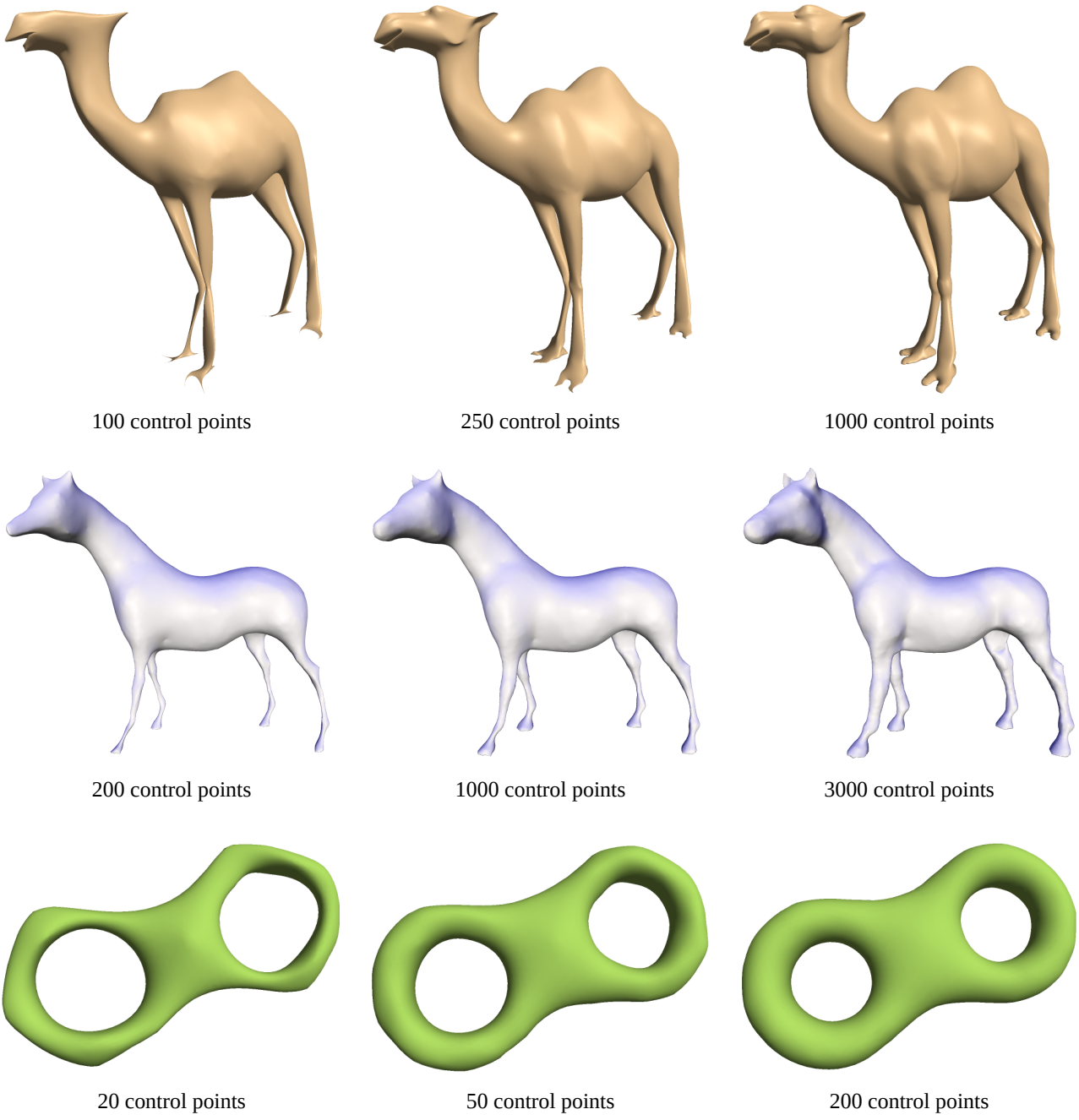


Figure 11. Examples of different LS-meshes. Each row displays LS-meshes computed using the same connectivity graph and a varying number of control points. Note that LS-meshes can have arbitrary topology, including genus greater than zero.

LS Meshes

OlgaSorkine　DanielCohen-Or
特拉维夫大学
sorkine@tau.ac.il
dcor@tau.ac.il

摘要

本文将介绍一种最小方差 mesh (Least-squares Meshes), 即在给定连接关系和控制点的前提下获得具有最小方差意义的网格. 给定的 mesh 包含一个任意连接关系的平面图以及一系列稀疏的控制点. 通过解一个稀疏线性方程组我们将会得到一个新的网格图. 这个线性系统不仅定义了由给定控制点所约束的曲面, 而且使顶点均匀地分布在曲面上---也就是说, 每一个顶点都位于尽可能接近其邻接顶点的几何重心. LS-meshes 是可见地平滑且均衡化控制顶点的. 我们将会看到曲面顶点的连接包含了曲面的几何信息且将会影响曲面重构的效果. 本文最后将会讨论 LS-Meshes 的应用, 如平滑化曲面, 编辑网格等.

一 简介

本文将会介绍 Least-squares Meshes: 由给定图连接关系和控制顶点所定义的具有最小方差意义的网格. 给定一个平面图及其顶点连接信息, 还有一系列稀疏的控制顶点, 我们可以通过解一个线性系统得到新的网格顶点. 这个线性系统不仅定义了靠近控制点的曲面, 而且是顶点均匀地分布于曲面上. 也就是说, 每一个顶点都位于尽可能接近其邻接顶点的几何重心. LS-meshes 是可见地平滑且均衡化控制顶点的.

本文还证明了通过合理地选取控制顶点, LS-mesh 可以有效地接近一个给定的网格. 图 1 显示了一个骆驼模型的关联矩阵和控制点, 图 2(a)显示了有 20000 个顶点的马模型, 在 2(b)中, 我们使用了相同的关联关系但点的个数只是其实其中的 1000 个. 放大图 2(d)说明了 LS-Mesh 的效果.

一般的散乱数据技术[7]一般接受一系列不具备结构的点,然后试图修正(如通过插值等方法)一个连续曲面使得满足特定的条件,例如光滑性. 为可视化或更层次的需求, 连续曲面通常接下来会被采样或三角面片化. 当一个隐函数满足了接近一系列的点[3,15,17], 他们就定义了一系列曲面所需要提取的函数. 然而, 相反地, LS-Mesh 方法直接地给出尽可能满足给定点的曲面的网格.

我们的工作十分接近于浮动凸插值方法[6], 这种方法适用于平面中. 我们在本节后面将会介绍到: 我们的方法和浮动插值方法均可以是顶点均匀地分布. 然而, 浮动凸插值方法中的控制点是硬约束, 而我们的方法中控制点是软约束—符合最

小二乘意义上的约束. 浮动凸插值方法适用于构造一个二维平面的几何网格，而我们的方法可应用于任何三维表面模型中. 特别地, LS-Mesh 方法可以构造包含任意给定连接关系的图(网格)，以及生成形状与属性为正的任何包含边界的表面.

另一个相关的工作是 variational subdivision schemes [11,12]，这个方法可应用于解决散乱数据的插值问题. Kobbelt [12] 使用了Gauss-Seidel 迭代使薄平面的能量值得到最小化.

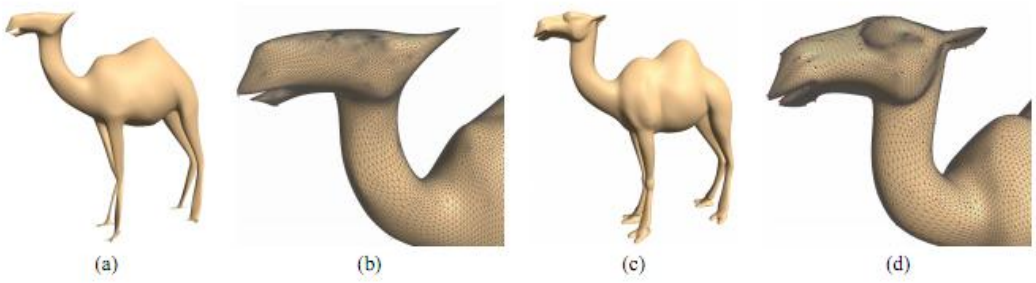


图1 : LS-Mesh: 由给定连接关系的图和一系列系数控制点所构造而成的网格. 在这个例子中，使用的例子是骆驼. (a) 由100个控制点形成的网格, (c) 由2000个控制点形成的网格. 连接图包含39074个顶点(没有任何位置信息). (b)和(d)是骆驼头部的方法图，控制点被标记为小点.

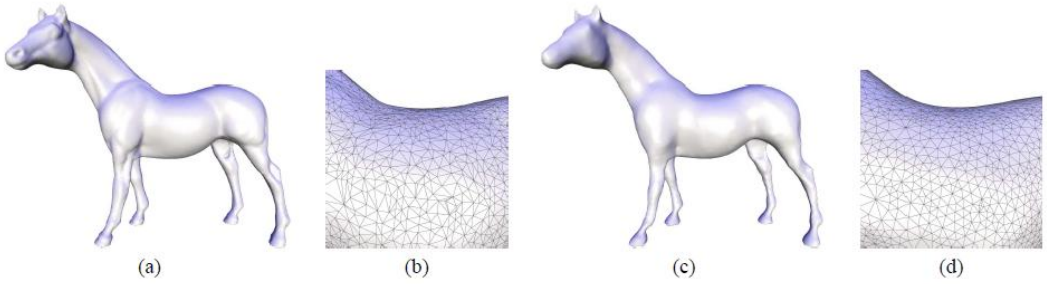


图2: 马的原始模型, 19851个顶点; (b) 原图连接关系的放大图; (c) 由1000个控制顶点形成的LS-Mesh; (d) LS-Mesh关联关系的放大图

这些迭代过程就是是网格平滑化的滤波器. 交错地使用Gauss-Seidel迭代和细划分可以将一个稀疏网格表格变成一个光滑的表面.

本文接下来的组织如下. 下一节将回顾一些LS-Mesh所需要的数学知识. 第3节将讨论LS-Mesh的性质. 第4节将介绍针对不同地表面获取LS-Mesh的策略. 第5节将说明解线型最小二乘的算法. 第6节讨论结果及应用. 第7节总结全文.

二 总览

令 G=(V,E)是一个给定的网格图, V={1,2,…,n}是其顶点集, E 是边集. 顶点 V_i (位置未知)为空间中的第 i 个顶点. 下面的方程定义了对于每一个顶点 V_i 的光滑性(类似于[6]):

$$\mathbf{v}_i - \frac{1}{d_i}\sum_{j:(i,j)\in E}\mathbf{v}_j = 0, \qquad (1)$$

其中 d_i 是顶点 i 的度数, 如果上面的方程被满足, 则顶点 i 位于其邻接点的中央.

Tutte[18]还证明了如果我们假定点 V_i 在一个 2D 平面中,上面的方程则定义了一个平面上一个有效的子图,图的边界在一个凸多边形内部. Floater[6]将结果拓展为该系统的结果使得所有的顶点在其邻接顶点的凸组合内. 在这里, 我们使用相似的方法解一个 3D 网格, 也就是说 V_i 在 R^3 中. 该线性系统可以写成矩阵的形式:

$$L\mathbf{x} = 0, \;\; L\mathbf{y} = 0, \;\; L\mathbf{z} = 0,$$

x,y,z 是 n 个顶点的 x,y,z 坐标构成的向量. L 是由下定义的 n*n 的矩阵:

$$L_{ij} = \begin{cases} 1 & i = j \\ -\frac{1}{d_i} & (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

L 就是著名的拉普拉斯算子[5,10,16], L 的秩为 n-k, 其中 k 是图 G 的连通分支数. 因为如果 G 是连通的, 那么 L 的秩是 n-1, 方程有 1 个基础解析.

没有给出任何几何信息的话,方程的解是毫无意义的. 如果给出 m 个顶点的三维位置信息,那么方程就会有一个有意义的解. 我们令:

$$\mathbf{v}_s = (x_s, y_s, z_s), \quad s \in C, \qquad (2)$$

其中 $C = \{s_1, s_2, ..., s_m\}$ 是控制顶点的下标. 我们的线性系统就变成了((n+m)*n)的矩形: Ax=b, 其中,

$$A = \begin{pmatrix} L \\ F \end{pmatrix}, \quad F_{ij} = \begin{cases} 1 & j = s_i \in C \\ 0 & otherwise \end{cases}$$

$$b_k = \begin{cases} 0 & k \le n \\ x_{s_{k-n}} & n < k \le n+m \end{cases}$$

图 3 是一个网格的控制顶点及其对应的系数矩阵. 通过添加一行或以上的数据使得 A 是一个满秩矩阵, 在最小二乘意义下解这个方程组, 亦即找到一个解 **x** 使得下式最小:

$$\|A\mathbf{x}-\mathbf{b}\|^2 = \|L\mathbf{x}\|^2 + \sum_{s\in C} |x_s - \mathbf{v}_s^{(x)}|^2. \qquad (3)$$

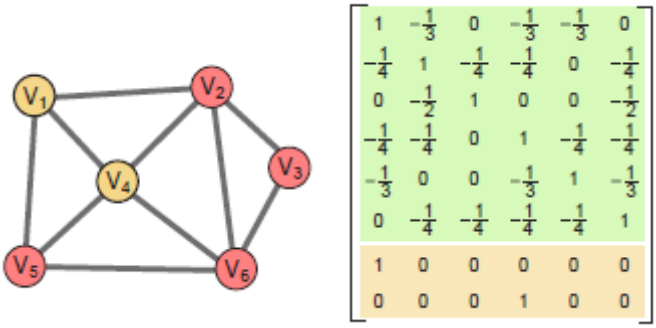


图 3: 构建系数矩阵的一个简单例子

这个方程唯一的解析解是 $\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}$,因为 A 是满秩的.

请注意在 Tutte 的 embedding 解中, 控制顶点的作用等同于边界顶点的作用. 他们通过给出几何信息限定了整个图的几何形状. 然而 Tutte 和 Floater 的边界系数矩阵解法和我们的方法最大的差异至于: 他们是假定这些控制顶点是位于一个固定不可变的位置, 而我们的方法是解得一个最小二乘意义上的解, 故控制顶点的位置是可变的. 我们保持了平面的光滑性, 并且方程的解对应于这些控制点的位置(包含多余的点).

为进一步理解上述的话, 我们可以对照图 4 及图 4 的说明:

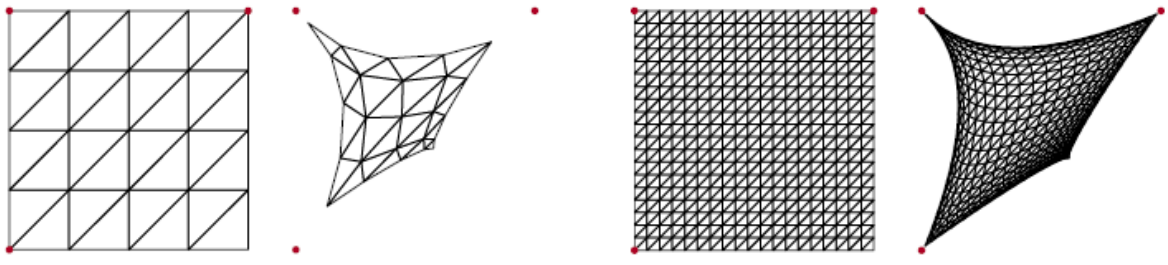


图 4: 矩形网格的 LS-Mesh. 第一和三列为该网格连通关系的可视化表示;点的位置是无意义的. 其中的 3 个位置边角的顶点用于生成 LS-Mesh, 他们的位置用小点表示出来. 请

注意 Ls-Mesh 是在最小二乘意义下的优化解，所以误差会不可避免地存在：控制顶点并不是插值形成，曲面的光滑性也不是被绝对地满足.

三　Connectivity Meshes

一个三维的 mesh 包含了其连接信息和几何位置信息，也就是说 mesh 顶点的三维坐标. 一个 connectivity mesh 是不包含任何形状的. 它由任意一个 mesh 移除顶点的位置信息后形成. 我们的工作是在 connectivity mesh 的基础上添加一些补充信息(部分顶点的位置信息). 我们的线性最小二乘系统重构了网格中的所有点，其中的控制点位置其邻接顶点的中心. 由于重构的系统同时作用域 connectivity mesh，所以新生成的网格也很接近于 connectivity mesh 原来定义的网格[9]. 在他们的工作中, Isenburg 等人还证明了单纯的 connectivity mesh 也是有一定的自然形状的(假定网格中的每一条边都有相同的长度). 他们使用了迭代优化的方法最小化了一个 mesh 膨胀系数的能量函数，进而使得曲面趋于平滑，且每一条边的长度趋于统一. 这个优化过程是非线性的，计算过程需要不断地迭代优化. 但不管怎么说，他们工作的主要贡献是一个单纯的 connectivity mesh 还是包含了一些几何位置信息.

介于此,LS-Mesh 被认为是非单纯的 connectivity mesh，因为它需要部分控制点的几何位置信息. 如果说 connectivity mesh 是要用于重构一个曲面的形状，那么控制顶点的几何信息就是用于更好地构建网格. 有趣的是，对于 connectivity 图形平滑性的规则化的优化过程正好是网格的拉普拉斯方程[9].

当一小部分的控制点被使用时, LS-mesh 是 connectivity 形状的一个重构—在最小二乘意义下的重构. 这一点在图 7 和图 5 中加以了说明.

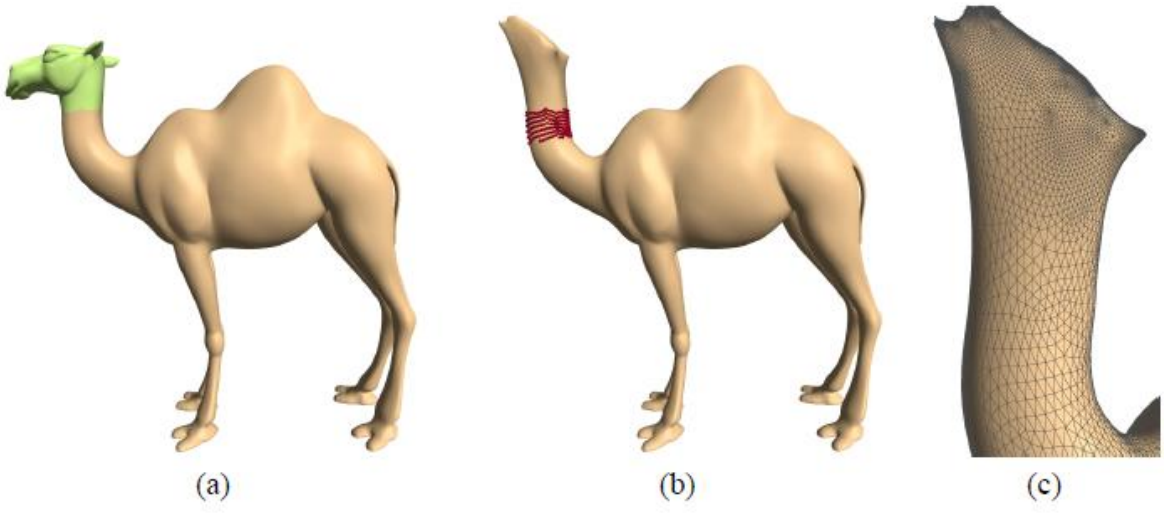


图 5. 用原始的连接关系信息重构骆驼模型的头部部分. 我们移除了骆驼的头部几何信息((a)中已标注). (b)，小洞附近的控制顶点被表述为小圆环. (c) 重构后的几何模型的局部放大图. 请注意头部的连接信息已包含了一些非平凡的形状，无需使用单一的控制点.

四 控制点的选取

LS-Mesh 可以用于重构一个给定的网格. 一系列的点需要设定以用于是 LS-Mesh 尽可能地接近原来的网格和使误差最小化. 因此, 不难想象, 这些控制顶点应该”有策略”地选取出来, 比如说选取一些有特征的点—诸如突出部分的某些点等等. 我们将要测试以下选取控制点的方法:

- 随机选取. 这是最快的方法. 然而, 如果原来的网格是高度不规则的, 包含了各种高频特征, 那么随机选取可能会损失很多的信息, 进而是结果并不理想(见图 6(a)).
- 逐个贪心选取. 这种方法每步选取一个新的控制点, 新的控制点是由已选中的点推导出来的最优的点. 所谓最优, 即新的点离其他原有点的距离误差最大. 这种方法比上一个方法要慢得多, 因为它每一步都要求解一个最小二乘方程组. 当然它是一个十分有效的方法, 它可以找到所有十分”重要”的点, 从而降低误差.
- 组合局部最大化方法. 使用这个策略时, 我们每 K 步计算一个最大化误差的控制点, 方法同上诉的贪心选取. 剩下的 K-1 步, 我们可以通过计算局部最大误差来求得. 也就是说, 我们按照宽度优先搜索顺序遍历这个网格, 从最近选取的控制点开始(这时可获得全局误差), 只要误差可降低, 我们就将这个点的邻接顶点标记为”隐藏”. 然后我们在剩下的没有被标记的顶点中选出一个能使误差最大化的顶点. 这个过程重复 K-1 此, 我们就可以得到一个新的 LS-Mesh. 这种方法比弹性算法要快, 因为其所需解的方程个数要小很多.

  作为例子, 我们可参见图 6. 虽然说贪心算法可以得到最好的控制顶点(即全局几何误差最小), 但组合局部优化的方法也做到了很相似的效果.

图 6. 使用不同的方法选取螺丝刀网格的控制顶点(共 27152 个顶点, 1000 个控制点). 下面一行的小点即为控制顶点. 在左边一列中,使用随机选取的方法. 中间一列使用了贪心的方法, 即每一个点的放置都是可最大化误差. 右边一列使用了组合式的方法, 每 31 步我们找

31 个局部最优的点放置在上面，然后重新计算 LS-Mesh 来获得下一个选择的控制点. 我们可以看到，随机的方法损失了很多有用信息，所以构造出来的网格也有十分大的误差. 而第二和第三个方法都能很好地保持了原来网格的信息，控制顶点都”很有策略”地落在了一些重要的位置，例如螺丝刀的凸痕,凹痕或边缘等.

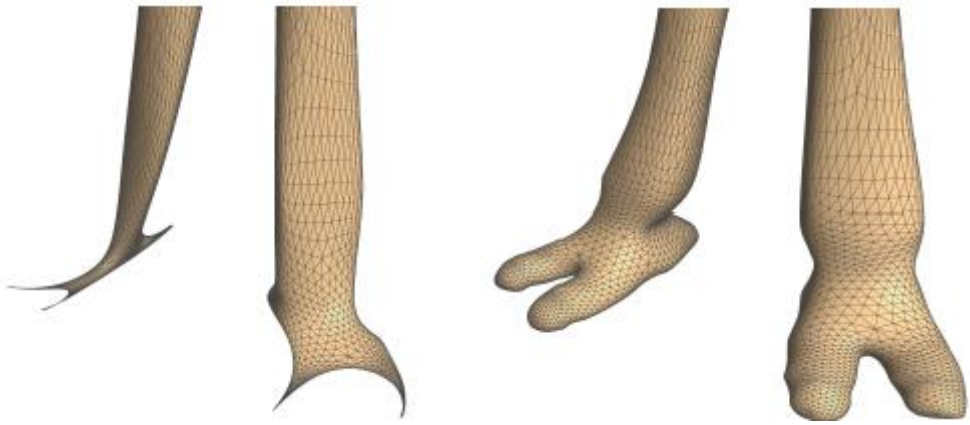


图 7. 骆驼模型足部的 LS-Mesh 局部放大图. 左边模型包含了 100 个控制点，后边模型包含了 2000 个控制点.

五 解方程组

LS-Mesh 需要接一个稀疏的线性最小二乘方程组来重构几何图形，也就是说最小化**||Ax-b||**. 我们使用最直接的方法解这个方程组$A^T Ax = A^T b$. 很明显，我们可以使用 LU 分解解这个方程组.

表格 1 记录了这个过程所需要用到的时间. 我的实验是在 2.4GHz 的奔腾 4 代计算机上进行的. 从表格中可以看出，整个过程耗时最多的步骤就是计算系数矩阵的分解，其他的如解方程组的时间是可以忽略不计的.

Model	# vertices	Factor	Solve	Total
Eight	2,718	0.085	0.004	0.097
Horse	19,851	0.900	0.032	0.996
Screwdriver	27,152	1.646	0.068	1.850
Camel	39,074	2.096	0.073	2.315

表格 1. 解线性最小二乘方程组的程序运行时间(单位: 秒). Factor 表示系数矩阵的分解，Solve 时间即为求出(x,y,z)的值的时间. 最后一列显示了计算 LS-Mesh 的总时间.

六 讨论

正如上文所述, LS-Mesh 可以得到与原图相近的图像. 当控制顶点的数目变得越少, 得到的图形就越接近于 connectivity mesh. 图 11 显示了控制点数目变化产生的效果. 但控制点的数目少到一定程度的时候, 得到图形的变形也很严重, 甚至完全看不出原图的模样. 但随着控制顶点数目的增加, 重构的图形会迅速地接近于原图. 给定一组赋有权重的顶点, LS-Mesh 可以得到更为优越的结果. 点的权重可以修改方程(2)可到, 即如下所示:

$$w_s \mathbf{v}_s = w_s(x_s, y_s, z_s).$$

因此, LS 系统的最小化能量值变为下面的形式:

$$\|A\mathbf{x} - \mathbf{b}\|^2 = \|L\mathbf{x}\|^2 + \sum_{s \in C} w_s^2 |x_s - \mathbf{v}_s^{(x)}|^2.$$

图 9 是一个平面图的例子, 在当中可以看到给定控制点的权重, LS mesh 的效果更好.

LS-Mesh 还能运用于填充曲面中的"洞穴". 这个与 Levy[13]中的方法类似. Levy 的方法适用于二维平面. 而通过解我们的线性系统, 我们可以知道洞附近有一些控制点, 从而洞中未知位置的顶点将会被推算出来, 从而起到了填充凹洞的作用. 例子见图 8.

另一个有趣的应用是模型的编辑. 通过制定控制顶点, 一个新的 LS-Mesh 就会马上生成. 然而我们的方法不同于[19]. 在这里网格的连通关系是给定的. 这就意味着我们的方法更有利于编辑已经存在的图形. 见图 10 马的例子, 我们移除掉其头部的控制顶点, 马上又会得到新的编辑后的网格. LS-Mesh 的重构速度是非常快的, 因为其系数矩阵分解的过程是一次性的(可以先于编辑计算出来的). 要得到编辑后 LS-Mesh, 我们仅需要进行一次线性方程组的回代操作即可. 见第 5 节中的运算时间分析.

研究 LS-Mesh 在图形压缩和三维数据传输中的运用也是很有意义的. 以往的工作也有做出相关的尝试[1,8,20]. 而在空间数据压缩方面, 本文的尝试从某种意义上接近于[10]的方法—两个方法都是假定原网格的连通关系是已知的.

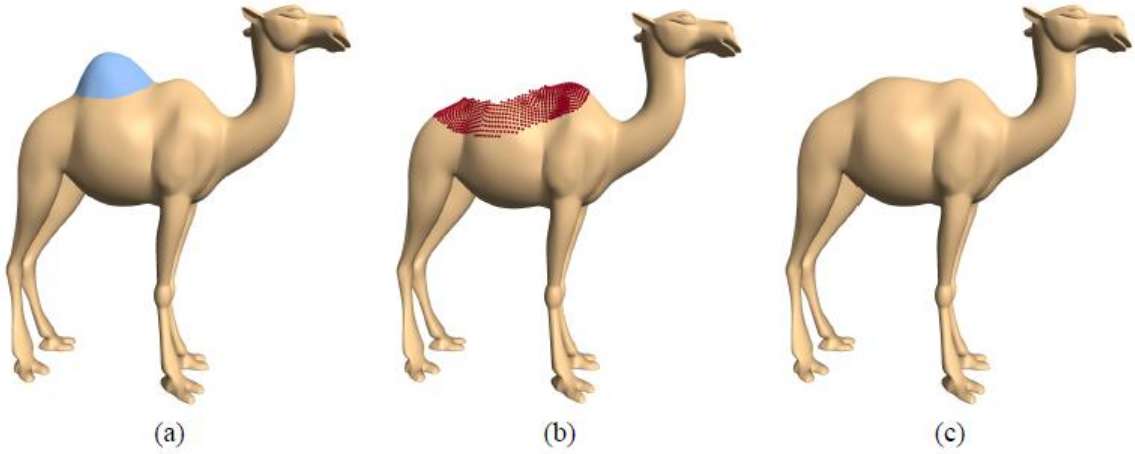


图 8. 使用 LS-Mesh 重构已经被移除的洞. (a) 骆驼模型的原型. 我们移除顶部的驼峰中的某些顶点, 然后使用某些控制顶点((b)中的小点)对其进行重构. (c)为重构的结果

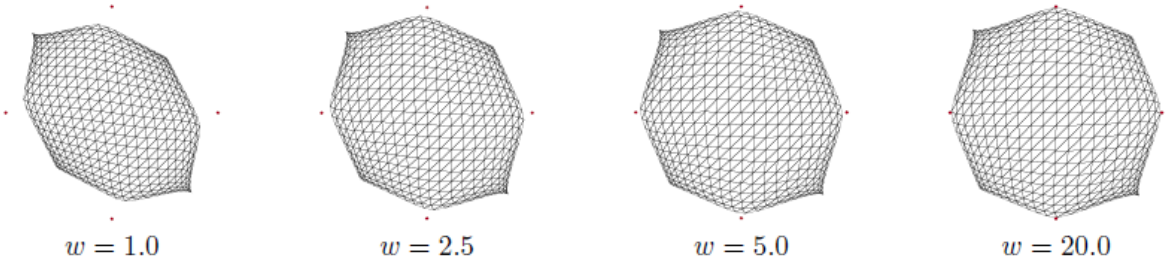


图 9. 使用不同的控制顶点权重约束进行 LS-Mesh 重构. 控制顶点在图中用小点表示. 随着控制顶点的增加, LS-Mesh 越来越接近于插值.

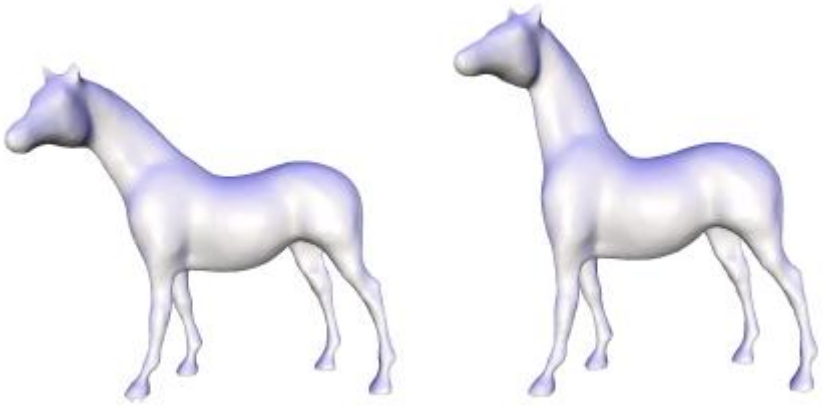


图 10. 编辑 LS-Mesh 的例子. 左侧为原图, 右侧为移除头部的控制顶点然后再重新重构说得结果

七 总结

LS-Mesh 概念上非常简单且很利于实现. 他们提供了一种针对给定连通关系的图和少量控制顶点的图的一种重构方法.几何图形的重构是通过解一个线性方程组完成的.

非常重要的一点, 方程(1)中的定义曲面光滑性只是由曲面的顶点连通关系确定的.

我们认为目前的研究开创了很多有趣的研究方向, 我们希望有其他感兴趣的人加入我们的研究中. 这些方向包括:

- 理解 LS 解和子曲面连通关系之间的关系.
- 使 LS Mesh 适用于一些带尖锐形状的图形中, 目前我们的方法只有在光滑曲面上才能取得比较好的效果, 我们希望改变这种劣势.
- 分析 LS-Mesh 的光滑性
- 在上述研究思路的基础上, 探索 LS-Mesh 应用于数据传输方面的运用.

致谢

We would like to thank Sivan Toledo for insightful discussions and Christian Rossl for proofreading. This work was supported in part by grants from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), by the Israeli Ministry of Science, by the German Israel Foundation (GIF) and by the EU research project Multi-resolution in Geometric Modelling (MINGLE)' under grant HPRN-CT-1999-00117. The screwdriver mesh is courtesy of Cyberware.

参考文献

[1] P. Alliez and M. Desbrun. Progressive compression for loss-less transmission of triangle meshes. In Proceedings of ACM
SIGGRAPH 2001, pages 198–205, 2001.

[2] M. Ben-Chen and C. Gotsman. On the optimality of spectral compression of meshes. Preprint, available online from http://www.cs.technion.ac.il/~gotsman/publications.html, 2003.

[3] J. F. Blinn. A generalization of algebraic surface drawing. ACM Transactions on Graphics, 1(3):235–256, July 1982.

[4] P. H. Chou and T. H. Meng. Vertex data compression through vector quantization. IEEE Transactions on Visualization and Computer Graphics, 8(4):373–382, 2002.

[5] M. Fiedler. Algebraic connectivity of graphs. Czech. Math. Journal, 23:298–305, 1973.

[6] M. S. Floater. Parametrization and smooth approximation of surface triangulations. Computer Aided Geometric Design, 14(3):231–250, 1997.

[7] R. Franke and G. M. Nielson. Scattered data interpolation and applications: A tutorial and survey. In H. Hagen and D. Roller, editors, Geometric Modelling, Methods and Applications, pages 131–160. Springer Verlag, 1991.

[8] H. Hoppe. Progressive meshes. In Proceedings of ACM SIGGRAPH 96, pages 99–108, August

1996.
[9] M. Isenburg, S. Gumhold, and C. Gotsman. Connectivity shapes. In Proceedings of IEEE Visualization 2001, pages 135–142, 2001.
[10] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In Proceedings of ACM SIGGRAPH 2000, pages 279–286, July 2000.
[11] L. Kobbelt. A variational approach to subdivision. Computer Aided Geometric Design, 13:743–761, 1996.
[12] L. Kobbelt. Discrete fairing and vibrational subdivision for freeform surface design. The Visual Computer, 16(3-4):142–158, 2000.
[13] B. L´ vy. Dual domain extrapolation. ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003), 22(3):364–369, 2003.
[14] H. P. Moreton and C. H. S´ quin. Functional optimization fore fair surface design. In Proceedings of ACM SIGGRAPH 92, pages 167–176. ACM Press, 1992.
[15] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. ACM Transacations on Graphics (Proceedings of ACM SIGGRAPH 2003), 22(3):463–470, 2003.
[16] G. Taubin. A signal processing approach to fair surface design. In Proceedings of ACM SIGGRAPH 95, pages 351–358, 1995.
[17] G. Turk and J. F. O'Brien. Modelling with implicit surfaces that interpolate. ACM Transactions on Graphics, 21(4):855–873, Oct. 2002.
[18] W. T. Tutte. How to draw a graph. Proc. London Mathematical Society, 13:743–768, 1963.
[19] J. Warren and H. Weimer. Subdivision Methods for Geometric Design: A Constructive Approach. Morgan Kaufmann Publishers Inc., 2001.
[20] J. C. Xia and A. Varshney. Dynamic view-dependent simplication for polygonal models. In Proceedings of IEEE Visualization '96 , pages 327–334., 1996.

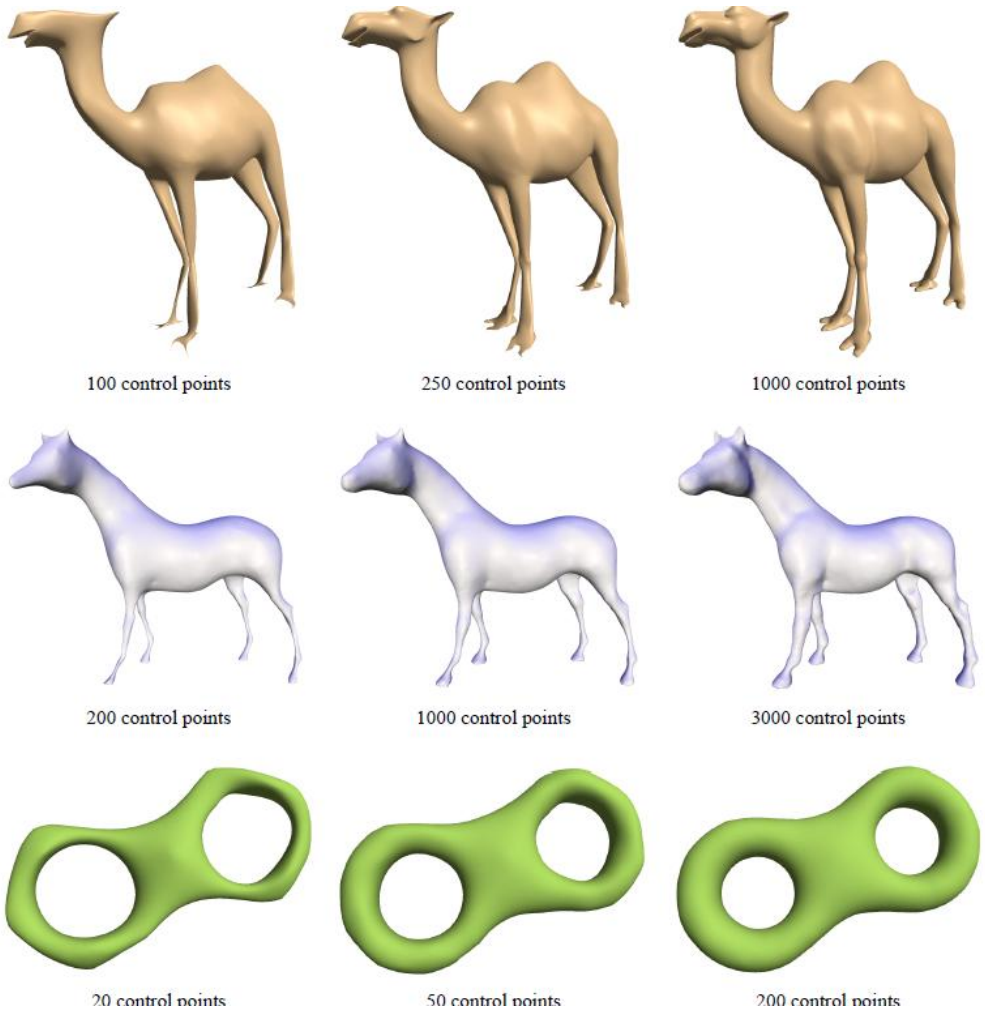


图 11. 其他 LS-Mesh 的例子. 请注意 LS-Mesh 适用于任何拓扑的曲面