



## 毕业设计(论文)

# 基于图像的轻量化3D树木建模

## 方法

姓 名：张德嘉

学 号：092792

所在院系：软件学院

学科门类：软件工程

学科专业：软件工程

指导教师：贾金原

二〇一三年五月

## 摘 要

树木建模一直是计算机图形学中一个极具挑战并且非常重要的研究课题。随着目前WebVR、WebGame、WebGIS等基于Web的应用发展迅速，为了适应网络的传输以及用户日益增长的对图形效果的追求，如何使树木建模轻量化而富有真实感就变得尤为重要。传统的3DSMAX、Maya等建模工具不仅耗费人力物力，而且输出的面片模型也体积庞大，不适合应用到Web领域。而诸如L-System等基于规则的树木建模又由于其规则性而使树木模型缺失了真实感，这又不能满足用户对效果的需求。怎样在真实感和轻量化之间进行权衡的问题亟待解决。

为了解决这个矛盾，本课题提出了一套高效、低成本，的分级轻量化树木建模方法。这里的分级轻量化体现为其对应用的适应性。即可以基于不同应用对轻量化的不同要求，在尽可能保证真实感的前提下进行轻量化，以产生最终符合要求的模型尺寸。这种可分级的轻量化树木建模方法还可以进一步被扩展为自动适应网络带宽条件或用户硬件条件而自动产生最匹配的轻量化树木模型的方法。

为了实现高效的分级轻量化建模方法，本文首先将PyrLK光流法进行基于仿射变换和反向追踪的改进，并且将其运用到三维重建的特征点匹配步骤中，以提高树木特征点的匹配率和稳定性。然后进行GPU加速的三维重建以得到高精度点云模型。接着本文运用三维体素泛洪和最小二乘线性拟合的方法对树木骨架和半径信息进行抽取，以适应树木生长规律的方法抽取出了准确的骨架。然后本文提出了根据应用对轻量化的需求等级，对骨架进行纵向和横向的合并，以减小骨架的尺寸来实现轻量化，从而更好地适应面向网络的应用的需求。最后本文还给出了一套完整的基于图像树木建模的质量评价，提出了还原度的概念来客观、量化地评价建模出的模型的还原度以及在轻量化过程中质量的丢失。

**关键词：**基于图像建模，树木建模，轻量化建模，三维重建，骨架抽取

## Abstract

Tree modeling has long been a challenging subject in computer graphics. As the Web-oriented applications(WebVR, WebGame, WebGIS, etc) develop rapidly and the persuit of graphics effect increases quickly, the lightweight and realism of tree modeling are badly needed nowadays. The traditional 3d modeling tools such as 3DSMAX and Maya are not only time and labour consuming, but it takes a large model size, which are not practical to Web apps. The rule-based modeling such as L-System can solve the size problem, but its output lacks realism, which is not tolerated by users. So the balance between realism and lightweight is a real problem which are eagerly demanded to solve.

In order to solve this problem, a high-efficiency, low-cost and lightweight-classified tree modeling method is proposed. Here the lightweight-classified means it can produce different lightweight levels of tree models. And to implement this goal, this method will reduce model size on the premise of not losing much realism. This method can also be applied furthur to automatically adapt to the bandwidth and hardware conditions of the client side.

For implementing the lightweight method, we first improve the traditional PyrLK optical flow method to support affine transformation and backward feature tracking, which can furthur be applied to do feature matching in gpu accelerated 3D reconstruction and improve the match ratio. Then we use flooding algorithm in 3D voxel model and least squares method to discover the tree skeleton and its radius information. According to the lightweight level the applications require, we reduce the model size by merging the branches vertically and horizontally respectively. At last we propose a modeling quality evaluation method, which will objectively and quantizedly evaluate the restore degree of the tree model.

**Key words:** Image-based Modeling, Tree Modeling, Light-weight Modeling, PyrLK  
Optical Flow, Skeleton Extraction

# 目录

第 1 章 引言 .....	1
1.1 背景介绍 .....	1
1.2 课题的主要工作 .....	1
第 2 章 树木建模的研究综述 .....	3
2.1 虚拟树木建模技术 .....	3
2.2 现实树木重建技术 .....	4
2.3 现有方法的不足 .....	5
2.4 本章小节 .....	6
第 3 章 基于图像的树木轻量化3D建模方法 .....	7
3.1 技术路线 .....	7
3.1.1 基于改进的PyrLK光流法的图像特征匹配 .....	7
3.1.2 三维重建 .....	7
3.1.3 基于三维体素泛洪与线性拟合的三维树木骨架抽取 .....	8
3.1.4 基于用户交互的模型改善与轻量化 .....	8
3.1.5 建模质量评估 .....	8
3.2 技术路线图 .....	9
第 4 章 支持旋转和反向追踪的改进型PyrLK光流法 .....	10
4.1 传统光流法简介 .....	10
4.2 PyrLK光流算法 .....	11
4.3 PyrLK光流算法的改进 .....	12
4.3.1 加入仿射变换 .....	12
4.3.2 反向追踪与中值过滤以提高鲁棒性 .....	13
4.4 支持仿射变换和提高鲁棒性的的PyrLK光流法 .....	15
4.5 本章小节 .....	17
第 5 章 基于三维体素泛洪与线性拟合的三维树木骨架抽取 .....	19
5.1 三维体素模型 .....	19
5.2 三维体素泛洪确定邻域范围 .....	20
5.3 通过最小二乘法线性拟合确定分支方向 .....	22
5.4 获取子枝长度和半径 .....	24

---

5.5 本章小节 .....	27
第 6 章 基于枝干合并的轻量化处理 .....	28
6.1 L-System的尝试 .....	28
6.1.1 L-System简介 .....	28
6.1.2 树木模型的参数化L-System规则抽取 .....	28
6.1.3 使用L-System进行树木轻量化建模遇到的问题 .....	29
6.2 基于枝干合并的树木分级轻量化 .....	29
6.2.1 枝干纵向合并 .....	30
6.2.2 枝干横向合并 .....	31
6.2.3 联合使用横纵向合并 .....	32
6.3 本章小节 .....	33
第 7 章 建模还原度 .....	34
7.1 图像序列信息量 .....	34
7.2 三维重建还原度 .....	36
7.3 骨架抽取还原度 .....	36
7.4 建模还原度计算 .....	37
7.5 本章小节 .....	37
第 8 章 基于用户交互的模型改善 .....	38
8.1 加载树木点云文件 .....	39
8.2 显示骨架模型 .....	40
8.3 显示点云模型 .....	41
8.4 显示体素模型 .....	42
8.5 节点编辑 .....	43
8.6 简化模型 .....	44
8.7 泛洪算法演示 .....	45
第 9 章 实验过程与分析 .....	46
9.1 实验环境 .....	46
9.2 实验结果与分析 .....	46
9.3 本章小节 .....	55
致 谢 .....	56
参考文献 .....	57

# 第1章 引言

## 1.1 背景介绍

在互联网飞速发展的今天，网络应用已经延伸到生活的方方面面。微博、人人网、在线购物、在线音乐等已经成为当今人们生活的一部分。面向Web的虚拟现实应用如WebVR、WebGame、WebGIS等也必然将成为虚拟现实发展的重要方向。树木作为自然界常见的事物，在各种虚拟现实的场景中出现的频率很高。然而树木形态各异，结构复杂，给3D建模带来了很大的难度。通常单棵树的数据量已经不小，对于构建一个树木的聚集场景(如森林)就更加庞大，这容易使得场景负荷变大而产生延迟。因此，树木建模的质量和效率将直接决定面向Web的虚拟现实应用的成败。

目前的树木的3D建模，主要是通过专业的3D建模工具(3DSMAX、Maya等)进行手工建模。这种建模方法对建模人员的要求较高，并且需要的时间长。而且这种方法通常最终生成的是面片信息，要表达一棵形态复杂的树木需要大量的顶点信息，导致最终生成的模型体积较大，对于需要大批树木的场景，负荷就会变得更大。

目前树木的轻量化建模，从最简单的基于分形，广告牌技术的建模到稍微复杂的基于规则的建模，都存在一个共同的问题，就是在轻量化的同时，很大程度上舍弃了模型的真实感和树木本身的形态特征。随着当今应用对真实度要求的升高，这类轻量化的建模方法已经不能完全满足需求。真实感与轻量化之间的权衡也成为了当今应用需要考虑的一个重要因素。

本课题基于以上的考虑，从基于图片对树木结构进行完整的恢复，到面向应用需要对真实感与轻量化进行人工控制，到最后模型重建质量的评估，给出了一套完整的解决方案。

## 1.2 课题的主要工作

本课题的主要工作有：

1. 对PyrLK光流法进行改进，使其适用于仿射变换的特征匹配，并增加反

向追踪提高算法鲁棒性。将其运用于三维重建算法中的特征点匹配步骤，使树木重建的模型更加准确和精细。

2. 提出了基于三维体素泛洪和线性拟合的树木骨架抽取方法。该方法区别于传统的3D瘦化骨架抽取方法，它只适用于具有分形结构的3D骨架，所以更能够准确的抽取树木的骨架。

3. 提出了基于用户交互对树木模型进行完善和轻量化，让最终的应用来决定其所需的树木模型，避免了主观的一味轻量化或一味追求真实感而带来的需求矛盾，将模型的成型延迟至具体的应用。

4. 提出了基于图像的树木重建质量评估方法，对于重建出来的树木模型进行了客观和量化地质量评估，计算出其还原度。

## 第2章 树木建模的研究综述

树木的建模和造型技术，是计算机图形领域颇具挑战性的研究方向之一。自上世纪六十年代起，大批国内外的研究工作者利用各种不同的方法和技术来构建树木的形态，大大地推进了树木建模技术的发展。目前，在树木的建模领域，主要存在两种不同的类别：虚拟树木建模和对真实世界的树木进行重建。

### 2.1 虚拟树木建模技术

虚拟树木建模主要是指所建模的树木对象并不是直接从现实生活中获取，而是根据一定的规律或生长机理模拟化地对树木进行建模，树木的结构等都是通过过程化的方法所生成，而非从点云中抽取。

一个虚拟树木建模的经典方法是由Lindenmayer于1968年提出的L-System<sup>[1]</sup>，它是一个“字符串重写系统”，后在90年代初，Prusinkiewicz与Lindenmayer一起将L-System规则系统用于描述树木的生长过程<sup>[2,3]</sup>。它用语法表达了植物的生长规则，加入了分枝角度、长度等信息，以便于植物的表达与生长。在此后以L-System为基础的研究工作中，部分研究人员用若干几何模型构建出植物的枝干，并且引入不同的参数来表示植物的生长。另一部分研究人员使用分形的方法来进行树木建模。近几年来，L-System的方法常常以用户勾画为引导，让用户在简便操作的前提下设定L-System的参数，从而构建出树木模型，Okabe在2006年，Anastacio在2009年<sup>[4]</sup>都在这方面作出了贡献。2010年，Hongchun Qu使用BHA自动机和马尔科夫方法自动化地从输入图像中提取了L-System规则，在自动化提取规则方面迈出了第一步。

虚拟树木建模的另一个研究领域是AMAP系统<sup>[5]</sup>。该系统通过观察植物的结构，对植物形式和结构获得定性地理解，然后定量的测量植物形态的数据。植物的生长有一定的随机性，通过概率分布和应用理论来表达随机过程。系统依靠强大的实地数据采集和分析模块，将植物的各项测量数据整合到植物数据库，植物的拓扑结构演化由马尔可夫过程进行分析获得。再通过模式识别方法分析数据中提取生长规则的类型来构造植物的几何形状，应用蒙特卡罗的方法仿真模拟植物的模型，再应用几何的方法来表达其形成规律，并由此制作模型参数表，最后在场景中生成植物的图形。

## 2.2 现实树木重建技术

现实树木重建是指从现实生活中通过照片，视频或者三维扫描，来获得树木的实际数据，通过一系列重建的方法来对树木的几何，物理信息进行复原的过程。

基于图像的树木建模技术，是以在现实中拍摄的树木的图片作为输入，然后根据图片附带的树木信息重建出树木结构的技术。2004年，Reche-Martinez以空间体素的形式重建出了照片中的树<sup>[6]</sup>。2007年，Neubert不仅近似的生成了空间体素，而且还进一步以3D粒子流的方法来模拟生成了细枝和枝干<sup>[7]</sup>。该方法将树木上的点看作吸引子，以物理的吸引力等概念重建出了树木的信息，十分新颖。图片同样可以用来抽取L-System规则，对于树冠密集的树木，Shlyakhter在2001年首次从图片中抽取出了可见部分的规则，然后运用L-System进行处理。对于树叶占据大部分区域的树木图像，香港科技大学学者权龙在2006年，用交互式勾画辅以稀疏3D重建的方法，很好的恢复了树木的信息<sup>[8]</sup>。而对于树干信息占据大部分区域的树木图像，谭平又分别在2007年和2008年，用自动化L-System和用户交互L-System的方法完成了树木的重建<sup>[9,10]</sup>。2010年，Luis D.Lopez等人提出了一种从稀疏图像序列重建无叶树木的方法。2011年，Chuan Li等人提出了一种从树木视频输入重建树木模型的方法<sup>[11]</sup>。这两种方法均先从图像中获取2D树木骨架，然后对2D树木骨架序列进行三维重建获得三维骨架。由于树木图像本身存在遮挡导致2D树木骨架无法准确抽取，从而影响3D树木骨架的生成，因此这两种方法还原度都比较低。

基于激光扫描仪的体素化模型生成技术，是一种更直接地现实树木重建技术。它利用激光的单色性好、方向性强、能量高、光束窄等特点，直接对树木进行激光扫描，从而得到非常密集的点云数据。大多数基于激光扫描仪的方法都将重点放在了恢复代表枝干的骨架上，因为扫描出的叶子的点云含有太多噪声，以致无法准确重建其信息。1999年，Lazarus和Verroust用生成树的边长来聚合点云，从而获取骨架<sup>[12]</sup>。Bucksch在2008年和2009年将点云分块到八叉树表示的格子，然后用相邻格子间的连线来模拟骨架的曲线<sup>[13]</sup>。2007年Xu用启发式的方法来从扫描的点云中重建树木的主干，然后再用人工合成的办法在其上添加细枝和叶子<sup>[14]</sup>。Côte在2009年同样用人工合成的方式去构建细枝和树叶，但是他对光照散步的合成是通过在扫描的时候进行采样<sup>[15]</sup>。

## 2.3 现有方法的不足

综上所述，基于L-System的方法虽然轻量化，但是其试图用少量的规则来刻画自然中本就不规则生长的复杂的树木，导致了真实感的缺乏，同时在一个复杂结构中抽取L-System本身也是带有人为主观性和二义性的。基于AMAP系统的树木建模，虽然其建模的还原度较高，但是它需要大量的数据采集和专业的植物学知识，这对于一般性的应用来说显得超负荷。对于目前的基于图像的建模技术，如何准确的进行三维重建和骨架恢复仍然是一个难点。而基于激光扫描仪的树木建模，设备的价格又太高昂，而且对于骨架的抽取仍有一定的二义性存在。

同济大学图形图像实验室基于此做了一些先备工作，对基于图像的树木轻量化建模提出了建设性的思想和方法。其采用 PyrLK光流法，改善了三维重建的特征点匹配步骤。并且运用基于空间反向投影的方法，重建树木点云。之后再利用邻域探索点云分布的方法对树木骨架进行抽取，并通过L-System方法对骨架进行了规则抽取，以达到轻量化。虽然该方法对以往的一些方法进行了完善和改进，但是仍然存在以下不足：

- **特征点匹配缺乏准确性和鲁棒性** 虽然PyrLK光流法代替SIFT特征点匹配，从一定程度上改善了树木的特征点匹配效果。但是由于无法捕捉特征点的旋转变换，使得其特征点匹配的丢失情况严重。而且由于匹配缺乏一个验证正确性的机制，导致出现了过多的错误匹配。
- **骨架抽取不准确、不完整** 由于骨架抽取步骤中邻域的扩展运用增加边长的方法，在寻找一个分支的时候，容易受到其他分支影响，并且由于增加边长的方法是“暴力”的，导致效率很低。而且该骨架抽取方法没有给出合理的半径和长度的估算公式，导致根本无法还原出完整树木骨架信息。
- **轻量化效果欠佳** 对于自然中真实感极强的树木，利用L-System并不能很好的迭代地搜索出相同的规则，这是因为树木的结构本就是多变和复杂的。运用参数化L-System的方法将产生大量的规则，而导致没有从根本上解决轻量化的需求。

为了解决这些问题，弥补现有方法的局限，本课题结合现有方法的优点，改进其不足，提出了一整套基于多张图像的树木轻量化建模的解决方案。该方法较好的综合了真实感与轻量化，产生了很好的效果。

## 2.4 本章小节

国内外学者们多年来通过各种不同的技术和思想对树木建模的发展作出了卓越共享，也为后续工作者的研究奠定了坚实的基础。本章首先简要介绍了两大类树木建模的领域，即虚拟树木建模与现实树木重建。然后通过总结这两大领域中的已有方法，从规则生成，植物学领域，三维重建，骨架抽取等角度思考了树木建模问题。在同济大学图形图像实验室的已有工作上，本文进行进一步的创新和完善，并提出了一套兼具理论性和实践性的基于图像树木轻量化建模方法。

## 第3章 基于图像的树木轻量化3D建模方法

### 3.1 技术路线

本文提出了一套完整的基于图像的树木轻量化3D建模的方法。该方法首先以树木图片序列作为输入，用经过改进的方法对树木进行三维重建，使三维重建得到的模型精确度和完整性都得以提高。然后再用基于空间方向迭代和步长探索的方法抽取树木的骨架，最终再基于用户交互对骨架进行改善与轻量化。

该技术路线旨在实现一个对建模设备和条件要求不高，适用于一般应用的方法。在方便和简单的基础上，尽可能多的加入自动化，并结合少量用户交互，以实现高效、精确的树木轻量化3D建模。

该方法的主要步骤如下：

#### 3.1.1 基于改进的PyrLK光流法的图像特征匹配

本文首先将著名的金字塔LK光流法(PyrLK)进行了改进和扩展，使本来以平面平移作为运动假设的PyrLK光流法扩展为以平面仿射变换为运动假设，这样便能在两帧以空间角度旋转而拍摄的图片中，捕捉到空间旋转变换的投影，提高了匹配的完整性和正确性。之后本文再将得到的匹配点进行反向匹配，只有在一定容错区间内的匹配点对才会被接纳，并且进行中值阈值，将邻域相似度小于中值以及反向匹配度小于中值的匹配对剔除，以提高匹配算法的鲁棒性。

#### 3.1.2 三维重建

特征匹配完成以后，本文使用了美国华盛顿大学西雅图分校Changchang Wu的可视化运动恢复工具VisualSFM来完成基于多张图片的树木三维重建。VisualSFM实现了SiftGPU(GPU加速) 和多核的捆集调整(Multicore Bundle Adjustment)，使得相机参数的恢复更加快速和精确。在这个步骤本文用经过改进的PyrLK光流法的匹配结果替换VisualSFM中的 SIFT特征点匹配文件，进一步地改进了相机参数恢复的准确度和可信度。

### 3.1.3 基于三维体素泛洪与线性拟合的三维树木骨架抽取

在完成了三维重建之后，将会得到一个比较完整的树木空间点云模型。本文根据该点云的空间分布，并结合树木自底向上的自然生长规律和分形的逻辑结构特征，在阈值范围内，进行三维的体素泛洪，同时向多个子方向进行迭代，不断增加步长来扩大邻域范围。在确定邻域以后将几个点数比例较大的方向作为分支方向，并用线性拟合的方法确定其精确的分支方向。在确定分支方向后，再根据点到线的距离和投影分别估算出半径和长度。同时在迭代过程中及时剔除已经形成枝干的点云，来加速泛洪算法的完成。最终获取到的骨架信息是含有父子关系的节点信息，相比起3DSMAX等手工工具导出的面片模型，这种逻辑结构的模型大大的减小了其尺寸，但是由于逻辑结构并没有多少丢失，所以极具真实感。并且这种结构更便于后续的处理和进一步轻量化。

### 3.1.4 基于用户交互的模型改善与轻量化

由前面方法所得到的树木三维骨架虽然已经是含有父子信息的树木逻辑结构，但是由于前面的步骤都是自动化生成，所得到的结果不可能100%地保证符合具体应用的需求。并且前面的骨架信息虽然比起用面片来表示树木模型已经大大的轻量化了，但是针对实际的应用，本文还可以根据用户的交互来合并分支，从而进一步对模型进行轻量化，以适用于轻量化要求更高的应用。

### 3.1.5 建模质量评估

对于一个通过建模获得的树木模型，如果没有一个客观的量化评价指标，就无法从客观的角度反馈树木模型的还原度和各个步骤算法的可行性。本文提出了建模还原度的概念，以模型重建的还原度来量化的表现建模的质量。该还原度分别计算输入图像序列的信息量、三维重建的还原度和骨架抽取的还原度。并将它们有机地结合起来形成了最后总的建模还原度。

## 3.2 技术路线图

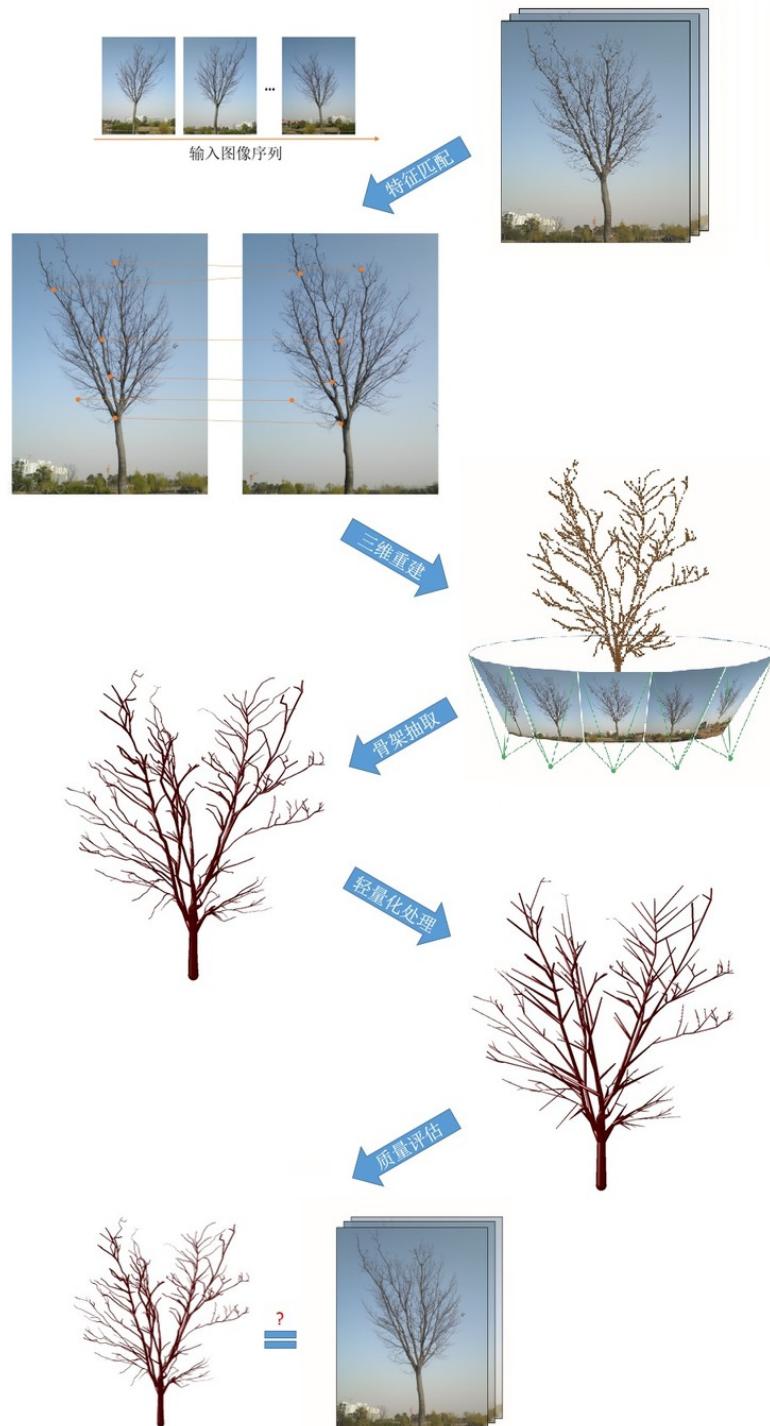


图 3.1 技术路线图

## 第4章 支持旋转和反向追踪的改进型PyrLK光流法

基于图像的树木建模第一步是三维重建，而三维重建的第一步则是特征点的匹配。所谓的特征点匹配，是在多张图片中找到空间同一个点在其上的投影位置，从而为三维重建的后续步骤提供数据支持。这里的特征点，本文选择了具有平移和旋转不变性的 Harris角点，以便用已有的方法快速找出图片中的特殊位置点。传统的PyrLK光流法，具有相当大的局限性，主要表现在：

- **不支持旋转** 传统的PyrLK光流法只支持特征点间的平移变换，从而导致其适用和推广受到了很大的局限。
- **缺乏验证机制** 没有给出一个对于匹配点是否合格的验证机制，导致匹配对的可信度下降。

为了突破PyrLK光流法的局限性，本文给出了基于仿射变换和反向追踪的PyrLK光流法，该方法很好的解决了以上两个问题，保证了其适用性和可信度。

### 4.1 传统光流法简介

光流的概念最早是由James Gibson提出的。1981年，Horn和Schunck创造性地将二维速度场和灰度联系起来，提出了一种有效的光流计算方法<sup>[16]</sup>。基于亮度不变的假设，图像灰度分布的变化由背景或目标的运动引起，背景或目标的灰度不随时间变化。在这种假设中，光流法通过目标和背景的不同速度来检测运动目标。

进一步说，将三维空间中的目标和场景对应于二维图像平面运动时，他们在二维图像平面的投影就形成了运动，这种运动以图像平面亮度模式表现出来的流动就称为光流(Optical Flow)。也就是说，光流是空间运动物体在观测成像面上对应像素点运动的瞬时速度，这个速度在图像中以每秒像素点的位移个数来衡量，它巧妙地运用2D的灰度变化来表征3D物体的位置和结构变化。而光流场(Optical Flow Field)就是所有光流点的集合，是一个2D瞬时速度场。光流场能够表征整个图像的位移变化，从而对3D运动目标进行检测和跟踪。

在光流法提出以后，很多学者对其进行了研究和改进，并且它们的方法各具特点，算法性能和运用场景各异。其中颇具代表性的是Lucas-Kanade局部平

滑法(LK光流法)<sup>[17]</sup>，它用基于微分的方法，利用时变图像灰度的时空微分来计算速度矢量，并且加以图像平滑处理，来进行光流跟踪。后来在2000年Jean-Yves又提出的基于图像金字塔实现的LK光流法，称为PyrLK光流法<sup>[18]</sup>。

## 4.2 PyrLK光流算法

假设图片上的像素点的值函数为 $I(x, y, t)$ ，表示坐标位于 $(x, y)$ 的像素点在时刻 $t$ 的像素值为 $I(x, y, t)$ 。那么经过 $\Delta t$ 时间后，像素值将变为 $I(x + \Delta x, y + \Delta y, t + \Delta t)$ 。有如下推导：

$$\begin{aligned} I(x + \Delta x, y + \Delta y, t + \Delta t) &= I(x, y, t) + \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t \\ \implies \frac{\partial I}{\partial x} \Delta x + \frac{\partial I}{\partial y} \Delta y + \frac{\partial I}{\partial t} \Delta t &= 0 \\ \implies \frac{\partial I}{\partial x} V_x + \frac{\partial I}{\partial y} V_y + \frac{\partial I}{\partial t} &= 0 \\ \implies I_x V_x + I_y V_y &= -I_t \end{aligned} \quad (4.1)$$

Lucas-Kanade光流法算法基于以上原理，并假设两帧图像之间发生的位移是微量的，而且在一个点的邻域内这个位移量是常数。这样可以对一个以 $p$ 点为中心的窗口内的像素点写出一个光流方程组，表征局部图像的运动向量 $(V_x, V_y)$ 需要满足以下方程组：

$$\left\{ \begin{array}{l} I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2) \\ \vdots \\ I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n) \end{array} \right. \quad (4.2)$$

这里的 $q_1, q_2, \dots, q_n$ 是局部窗口内的点， $I_x(q_i), I_y(q_i), I_t(q_i)$ 是图片 $I$ 对 $x, y, t$ 的偏导函数在 $q_i$ 处的值。将其写为矩阵形式得：

$$A = \begin{pmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{pmatrix}, \quad v = \begin{pmatrix} V_x \\ V_y \end{pmatrix}, \quad b = \begin{pmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{pmatrix} \quad (4.3)$$

这个方程组的方程个数远远多于未知数，所以 $A$ 是过约束的，LK光流法运用最小二乘法来求解出其光流速度。最小二乘法可以参考5.3或查阅相关资料。

LK光流法虽然比较直观，但是存在一个问题，由于能够探测到的运动块的大小和所选窗口的大小呈正相关，为了能够捕捉到大像素块的运动，需要将窗口大小相应调大。但是窗口大小越大，速度就需要在越大的邻域内保持稳定，就越不符合光流在小范围内稳定的假设。基于金字塔的LK光流法是Lucas-Kanada方法的一种改进版实现，它解决了窗口大小的与大块运动捕捉的矛盾。其具体思想如下：

设 $I$ 和 $J$ 是两张灰度图片， $I(x)$ 和 $J(x)$ 分别表示图片 $I$ 和 $J$ 在位置 $(x, y)$ 处的灰度值。现考虑图片 $I$ 上的一点 $\mathbf{u} = (u_x, u_y)$ ，特征追踪的目标就是找到图片 $J$ 上的一点 $\mathbf{v} = \mathbf{u} + \mathbf{d} = (u_x + d_x, u_y + d_y)$ ，使得 $I(\mathbf{u})$ 和 $J(\mathbf{v})$ “相似”。其中向量 $\mathbf{d} = (d_x, d_y)$ 表示图片在点 $\mathbf{x}$ 处的光流速度。下面来定义基于邻域的相似，设 $\omega_x$ 和 $\omega_y$ 是两个整数，将使得下面式子最小化的向量 $\mathbf{d}$ 定义为光流速度：

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x, y) - J(x + d_x, y + d_y))^2. \quad (4.4)$$

其中邻域窗口大小为 $(2\omega_x + 1) \times (2\omega_y + 1)$ 。式子的含义为寻找向量 $\mathbf{d}$ 使得 $\mathbf{u}$ 和 $\mathbf{v}$ 在邻域窗口大小内的差异最小化。

然后该方法将图像金字塔化，即将原图像作为最高分辨率层，逐步降低图像的分辨率，并作为新的一层，加入到LK光流法的迭代序列。通过这样多分辨率图层，使得邻域窗口的大小在低分辨率图像对应的区域可以映射到高分辨率图像的更大的像素区域，从而支持了大块运动。

## 4.3 PyrLK光流算法的改进

### 4.3.1 加入仿射变换

对于PyrLK光流算法，已经能够很好的解决几乎任何像素块大小由平移主导的匹配。然而，这并不足以完美地解决树木上点的匹配问题。因为相邻的两帧图像要求在空间形成一定的夹角进行拍摄，这样在两帧图像上，也一定会产生由空间旋转投影过后带来的平面旋转。而这样的变换在PyrLK光流算法里是无法解决的，因为PyrLK只是简单的将点的匹配依赖于点的平移。所以，有必要对PyrLK光流算法进行由平移变换到放射变换的扩展。

假设两个点的匹配满足仿射矩阵 $A$ ，那么有：

$$\begin{pmatrix} \Delta x' \\ \Delta y' \\ 0 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \\ 1 \end{pmatrix} \quad (4.5)$$

将其代入式4.1得：

$$(a_{11} \ a_{12} \ a_{13} \ a_{21} \ a_{22} \ a_{23}) \cdot \begin{pmatrix} \frac{\partial I}{\partial x} \Delta x \\ \frac{\partial I}{\partial x} \Delta y \\ \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \Delta x \\ \frac{\partial I}{\partial y} \Delta y \\ \frac{\partial I}{\partial y} \end{pmatrix} = -I_t \quad (4.6)$$

运用最小二乘法可以得到A的解。

将PyrLK中的定义式4.4稍作修改可使得其支持仿射变换：

$$Let \quad \mathbf{a}_1 = (a_{11}, a_{12}, a_{13}) \quad \mathbf{a}_2 = (a_{21}, a_{22}, a_{23}) \quad \mathbf{b} = (d_x, d_y, 1)$$

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x, y) - J(x + \mathbf{a}_1 \cdot \mathbf{b}, y + \mathbf{a}_2 \cdot \mathbf{b}))^2. \quad (4.7)$$

### 4.3.2 反向追踪与中值过滤以提高鲁棒性

本文前面几个小节一直在探讨如何改进和完善匹配的方法，从而提高精度和匹配可信度。然而，这其中有一个问题，单方向的去追踪匹配点是否就能确定该两个匹配点真正的匹配呢？其实不然，要确定两个点完全符合之前算法描述的特点，还需要反向进行检查，看v和v之间的匹配是否是双向和可逆的。换句话说，本文之前定义的“相似”其实是单方面的u相似于 v，而 v是否相似于 u还不得而知。因此，考虑到算法的完整性和鲁棒性，有必要进行反向的匹配来确定它们完全匹配。或者退一步，给出一个容错的区间，定义当差异度小于阈值时，两个点“相似”。基于此出发点，本文提出了基于反向追踪来对匹配对进行验证的思想。

另外，对于任何的匹配算法，不可能真正地达到完美和全部匹配。所以对于用金字塔LK光流法进行匹配过后，应该进行一定地筛选措施，以除去一些

“边缘化”的匹配，使得所有匹配对都能有更高的可信度，而不是完全依赖于算法的实际表现情况。本文提出了基于中值过滤的筛选方法，该方法将对两个步骤进行中值滤值：

- **邻域相似度中值过滤:** 在实施过光流算法以后，对于每两个匹配的点，计算其归一化相关系数，作为其相似度。归一化相关系数的计算式如下：

$$R(x, y) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} T'(x', y')^2 \cdot \sum_{x', y'} I'(x + x', y + y')^2}} \quad (4.8)$$

然后计算所有匹配对的相似度中值，对于低于这个中值的点对，本文将视其为不满足可信度的点对而剔除掉。

- **反向追踪差异度中值过滤:** 在进行反向追踪过后，每个点对将得到一个差异值，表示从后一帧图片的点到反向追踪得到的点与前一帧图像的对应点的空间位置差。本文对于反向追踪差异度低于中值的点对进行剔除，视其为不满足可信度的点对。

经过反向追踪和中值过滤还剩下的匹配点对，本文视为可信的匹配点对，该步骤对于提高算法的鲁棒性尤为重要。算法1给出了反向追踪和中值过滤的伪代码：

---

#### Algorithm 1 反向追踪&中值过滤

---

**Input:** 反向追踪差异度数组  $\mathbb{B}[1..n]$ , 邻域相似度数组  $\mathbb{S}[1..n]$

**Input:** 源匹配点对数组  $\mathbb{P}$

**Output:** 过滤后匹配点对数组  $\mathbb{Q}$

```

1: 初始化反向追踪差异度中值  $mid_{back}$ , 邻域相似度中值  $mid_{similar}$ 
2: SortArray( $\mathbb{B}[1..n]$ )
3: SortArray( $\mathbb{S}[1..n]$ )
4:  $mid = (n + 1)/2$ 
5:  $mid_{back} = \mathbb{B}[mid], mid_{similar} = \mathbb{S}[mid]$ 
6: for  $i = 1$  to  $n$  do
7:   if  $\mathbb{B}[i] < mid_{back} \cap \mathbb{S}[i] > mid_{similar}$  then
8:      $\mathbb{Q}.AddMatch(\mathbb{P}[i])$ 
9:   end if
10: end for
11: return  $\mathbb{Q}$ 

```

---

## 4.4 支持仿射变换和提高鲁棒性的的PyrLK光流法

经过仿射变换支持和提高鲁棒性的PyrLK光流法比起传统的PyrLK光流法，其适应性和稳定性都有大大的提升。其主要表现在：

- **对旋转的支持:** 加入了仿射变换意味着新的匹配算法能够很好的适应从三维场景旋转而投影形成的二维的旋转，而本文对树木的拍摄每一帧都是在上一帧的基础上进行三维的空间旋转，所以对旋转的支持可以大大提升匹配的准确性。
- **匹配验证:** 对于每一对匹配点，本文运用归一化相关系数对它们的邻域的相似性做进一步验证，只有通过相似匹配验证的点对，本文才认为是真正相似的点对。
- **剔除不稳定匹配:** 单方向的匹配不能确认两个点真正的匹配，所以本文的反向追踪可以对匹配进行双向的确认，从而剔除不满足反向追踪匹配的点对。以进一步提高算法的鲁棒性。

图4.1展示了传统的PyrLK光流法与本文的改进版本的对比。这里的图片用了不同的颜色通道来表示，其中绿色通道代表源图片，紫色通道代表目的图片，用红色的线段表示图片间点的匹配关系。从图4.1(a)中可以发现，传统的PyrLK光流算法几乎没有任何的验证措施，所以导致其出现了大量的错误匹配。比如图中远处的树林，由于其离观察原点比较远，所以视角的转动会导致其运动位移较大，然而图4.1(a)中所示远处森里的光流值并不大，还有图中近处的树木样本的匹配也非常不准确，有相当一部分点匹配到了背景上面去，这必然会使重建过程中由于错误信息而重建出错误的点云。而图4.1(b)总，无论是远处大位移的树林还是近处的树木枝条，光流值的大小都比较合适，而且匹配度是相当高的。可见改进版本比传统版本对于树木特征点的匹配有明显的提高，由于图片特征点匹配是重建步骤中的第一步，所以准确和稳定的匹配对于后期的重建工作至关重要，它将使得后期的工作事半功倍。



(a) 传统PyrLK光流法



(b) 改进的PyrLK光流法

图 4.1 支持仿射变换和提高鲁棒性的PyrLK光流法与传统光流法对比图

算法2给出了经过添加仿射变换支持和提高鲁棒性的PyrLK光流法算法的伪代码：

---

**Algorithm 2** 支持仿射变换和容错机制的PyrLK光流法
 

---

**Input:** 图像 $I, J$ ,图像 $I$ 中的点 $\mathbf{u}$

**Output:** 图像 $J$ 中对应点 $\mathbf{v}$

```

1: 构建图像 $I$ 和 $J$ 的金字塔表示:  $\{I^L\}_{L=0,\dots,L_m}$  和  $\{J^L\}_{L=0,\dots,L_m}$ 
2: 初始化金字塔估计值:  $g^{L_m} = (g_x^{L_m}, g_y^{L_m}) = (0, 0)$ 
3: for  $L = L_m$  to 0 with step of -1 do
4:   定位图像 $I^L$ 上的点 $\mathbf{u}^L$ :  $\mathbf{u}^L = (u_x, u_y) = \mathbf{u}/2^L$ 
5:   设  $\mathbf{a}_1 = (a_{11}, a_{12}, a_{13})$     $\mathbf{a}_2 = (a_{21}, a_{22}, a_{23})$     $\mathbf{b} = (d_x^L, d_y^L, 1)$ 
6:   定义相似度:
    
$$\epsilon(\mathbf{d}^L) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x, y) - J(x + \mathbf{a}_1 \cdot \mathbf{b}, y + \mathbf{a}_2 \cdot \mathbf{b}))^2.$$

7:   最小二乘法估计出 $d^L$ , 使得 $\epsilon$ 达到最小值
8:    $L-1$ 层金字塔估计值:  $g^{L-1} = 2(g^L + d^L)$ 
9: end for
10: 最终光流向量:  $\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0$ 
11:  $\mathbf{v} = \mathbf{u} + \mathbf{d}$ 
12: if BackwardTrack( $\mathbf{v}$ ) = true then
13:   if MedianFilter( $\mathbf{v}$ ) = true then
14:     return  $\mathbf{v}$ 
15:   else
16:     return NULL
17:   end if
18: else
19:   return NULL
20: end if

```

---

## 4.5 本章小节

本章首先介绍了光流法的由来, 进而分析了LK光流法的思想和特点。然后对于基于图像金字塔的PyrLK光流法, 它采用多分辨率图像层来克服传统LK光

流法无法捕捉图像中大块运动的缺陷。但是受缚于LK光流法的前提，该匹配算法只支持特征点的平移运动，由此带来很大的束缚性。而且PyrLK光流法并没有对匹配的正确性进行验证，从而导致了很多错误的匹配对。基于这两方面的考虑，本章接着提出了支持特征点旋转变换以及反向追踪的PyrLK光流法，这两个改进从根本上解决了PyrLK光流法的局限性，使PyrLK光流法的适用性和正确性都大大的提升。本章的最后还给出了改进方法与传统方法的实验效果对比图。

## 第5章 基于三维体素泛洪与线性拟合的三维树木骨架抽取

在获取了精确的点云模型之后，出于后续轻量化的考虑，需要将模型的存储方式由密集的点云转化为逻辑的父子结构。用树形的数据结构来表达现实的树结构，这是很自然的想法，相对于面片结构，树形结构也是一种更为轻量化的存储方式。每个节点表示树枝的起点，存储着该节点的空间位置，半径和该节点的父子枝信息以及兄弟信息。一个节点和它的一个子节点形成一个空间线段，若干空间线段组成一条连续的树枝。

本文从树的生长规律入手，从根节点往子节点生长。生长的依据则为当前节点所在邻域内的空间点云分布，节点邻域大小由步长来控制，步长会探索式地递增，直到达到了增长的阈值，邻域大小才确定下来。然后从其点云分布拟合出各个分支的方向，从而生长出新的子节点，并递归地生长下去直到点云的边界。

### 5.1 三维体素模型

前面三维重建步骤得到的结果是一个点云模型，该模型中的点数量庞大，不适于后续的邻域搜索，因此我们需要对点云进行体素化处理。所谓体素化，就是将点云占据的空间划分成一个个的小立方体，每一个立方体称之为一个体素。

在将点云模型转化为体素模型以后，对于点云的邻域搜索便转化为了对于空间临近体素的搜索，体素的位置就反映了点集的位置，因此不用每次搜索都遍历整个点云，而是只用将步长范围体素中的点集遍历即可。由于体素是我们处理的基本单位，所以体素的大小也直接决定了体素模型的精度，因此，在确保非空体素的空间连续性和效率允许的基础上，本文建议让体素尽可能的小，以保证模型的精度。将点云模型转换为体素模型的伪代码在算法3中给出。图5.1形象地展示了将空间体素化以后树木的点云模型在体素块中的分布情况。

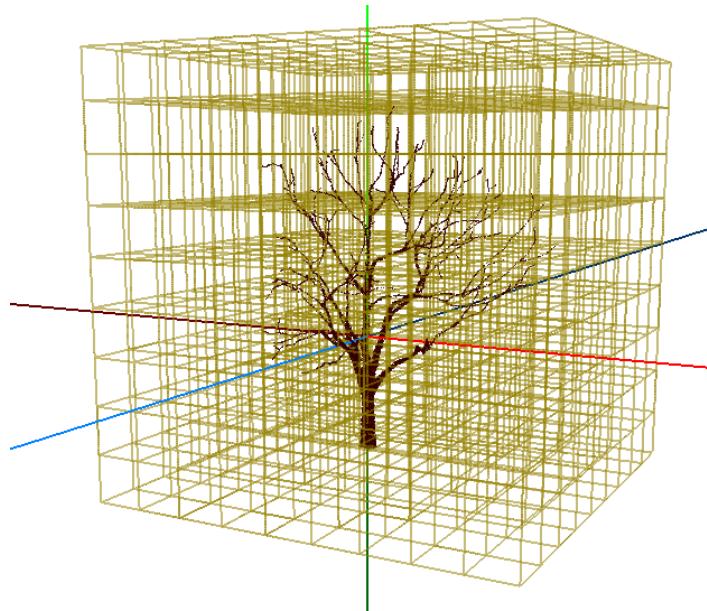


图 5.1 三维体素模型

**Algorithm 3** 点云模型体素化**Input:** 点云模型  $M$ **Input:** 体素维度  $d$ **Output:** 三维体素数组  $\mathbb{V}[1..d, 1..d, 1..d]$ 

- 1: 初始化点云边界值  $X_{max} = Y_{max} = Z_{max} = MIN\_FLOAT, X_{min} = Y_{min} = Z_{min} = MAX\_FLOAT$
- 2: **for all** 空间点  $P(P_x, P_y, P_z) \in M$  **do**
- 3:      $CheckBoundary(P)$
- 4: **end for**
- 5: **for all** 空间点  $P(P_x, P_y, P_z) \in M$  **do**
- 6:      $V_x = \frac{P_x - X_{min}}{X_{max} - X_{min}} \cdot d$
- 7:      $V_y = \frac{P_y - Y_{min}}{Y_{max} - Y_{min}} \cdot d$
- 8:      $V_z = \frac{P_z - Z_{min}}{Z_{max} - Z_{min}} \cdot d$
- 9:      $\mathbb{V}[V_x, V_y, V_z] = \mathbb{V}[V_x, V_y, V_z] \cup \{P\}$
- 10: **end for**

## 5.2 三维体素泛洪确定邻域范围

在确定了三维体素模型以后，便需要从根到叶，自底向上地对树的骨架结

构进行生长。生长的依据是已经得到的体素模型，将体素模型中点的分布作用于骨架的分支，便可以张成骨架模型。

具体方法是将根节点置为当前节点，对其进行三维泛洪，首先对其相邻的26个体素进行泛洪，若体素不为空，则将其加入邻域范围，若为空，则停止向该方向进行迭代。同时将加入邻域范围的体素置为无效，表示其已经参与了泛洪，不再参与骨架的重建，这样不仅可以对算法的结束有一个很好的约束条件，同时也可减少重复处理的次数，加快算法的完成。然后进行下一次迭代，对新加入的体素进行26方向的泛洪，并把有效的体素加入到邻域范围。接着比较两次迭代体素增加的比例，如果低于设置的阈值，则停止迭代，当前的邻域范围即为三维泛洪得到的当前节点的邻域范围。

图5.2展示了三维体素泛洪确定邻域的步骤，三张图都延空间z轴正向投影到2D平面。5.2(a)为其初始状态，即邻域范围为当前体素。其中橙色的区域表示邻域范围，蓝色的区域表示未探索区域，灰色区域表示空的体素，而绿色区域表示已经在之前的枝干邻域。5.2(b)表示体素泛洪经过一次迭代以后的状态，因为体素泛洪只会对与当前邻域范围相邻的未探索区域(蓝色方块)进行扩展，所以5.2(a)只会向黄色箭头指向的体素进行扩展，从而得到5.2(b)。在得到新的邻域后，首先会计算所新增的点的数量与之前的数量的比值有没有低于阈值，如果低于阈值，则停止邻域的扩张。最后将得到5.2(c)中的邻域范围。

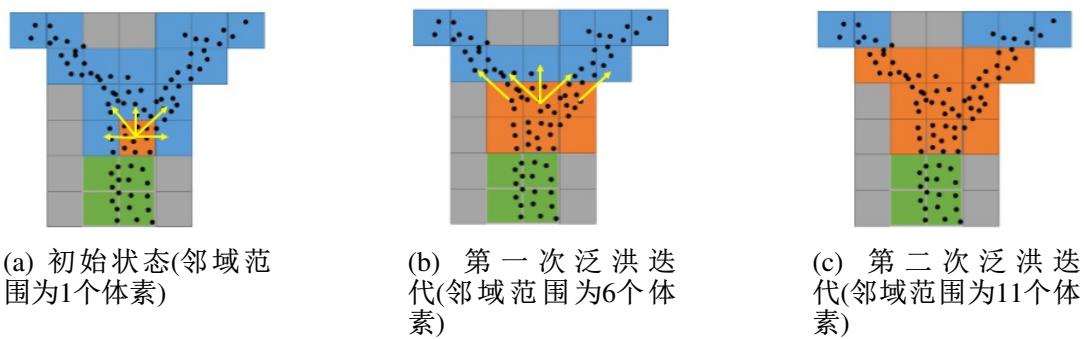


图 5.2 体素泛洪示意图

三维体素泛洪确定邻域范围算法的伪代码在算法4中给出。

**Algorithm 4** 三维体素泛洪确定邻域范围

**Input:** 当前体素 $C$ , 三维体素数组 $\mathbb{V}[1..d, 1..d, 1..d]$ , 泛洪方向数组 $\mathbb{D}[1..26]$ , 邻域范围增长比例阈值 $\lambda$

**Output:** 邻域范围内体素集合 $\mathbb{S}$

```

1: 初始化单次迭代新增体素集合 $\mathbb{S}'$ 
2:  $\mathbb{S}'.AddVoxel$ (根节点所在体素)
3: for all 泛洪方向 $Direction \in \mathbb{D}$  do
4:    $NewIndex = C.Index + Direction$ 
5:    $NewVoxel = \mathbb{V}[NewIndex.x, NewIndex.y, NewIndex.z]$ 
6:   if  $NewVoxel$ 非空 $\cap$   $NewVoxel$ 有效 then
7:      $\mathbb{S}'.AddVoxel(NewVoxel)$ 
8:   end if
9: end for
10: 体素增长比例 $\mu = \frac{\mathbb{S}'.VoxelCount}{\mathbb{S}.VoxelCount}$ 
11: if  $\mu > \lambda$  then
12:   for all  $voxel \in \mathbb{S}'$  do
13:     把 $voxel$ 作为当前体素进行递归调用
14:   end for
15: end if
16: return  $\mathbb{S}$ 

```

在进行三维泛洪的时候, 可以编程实现26个方向迭代过程的并行化, 以提高算法的效率。

### 5.3 通过最小二乘法线性拟合确定分支方向

当得到邻域范围以后, 便得到了邻域内体素在基于当前节点26个方向上的密度分布, 而每个体素内又包含着若干的点, 因此等于是得到了在当前节点邻域内的点云分布情况。接下来的工作就是怎样从各个方向的点云的分布情况抽取出核心的骨架。本文应用线性拟合的方法来从密集的点中抽取出一条直线, 作为该部分的骨架方向。

该方法首先要剔除掉那些点云密度很小的方向, 以免每个节点都朝各个方向长出一些细碎的枝条。因为这些细碎的枝条就算在此步中不剔除, 到后续的轻量化的时候也不容许它们的存在。

然后对于剩下的若干方向 $d_1, d_2 \dots d_k$ , 每个方向都对应着树木的一个骨架。在处理某个方向 $d_i$ 时, 将其包含的体素中的所有点抽取出来, 得到一个密集的点集 $S_i$ 。然后采用待定方程的办法, 设直线方程为:

$$\mathbf{x} = \mathbf{x}_0 + \mathbf{d}t, \quad (t \in [0, \infty)) \quad (5.1)$$

其中 $\mathbf{x}_0$ 是当前节点的坐标,  $\mathbf{d}$ 是待拟合的直线方向。我们假设点集 $S_i$ 中的点 $P_1, P_2, \dots, P_m$ 都在直线上, 则可以得到以下方程组:

$$\begin{cases} a_{11}d_x + a_{12}d_y + a_{13}d_z = b_1 \\ a_{21}d_x + a_{22}d_y + a_{23}d_z = b_2 \\ \dots \\ a_{n1}d_x + a_{n2}d_y + a_{n3}d_z = b_n \end{cases} \quad (5.2)$$

其中具体数值未给出, 注意这里的 $n = 3m$ , 因为每个点 $P$ 可以提供三个方向的方程式。在这个方程组中, 令

$$\mathbf{U} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} \end{pmatrix}, \quad \mathbf{d} = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{pmatrix}$$

在实践中, 由于筛选方向上的点数较多且发散分布, 由线性代数的理论知,  $\mathbf{U}$ 是过约束的, 即 $n > r$ , 其中 $r$ 是矩阵 $\mathbf{U}$ 的秩。这种情况下没有标准的解, 只能找到使误差最小的向量 $\mathbf{d}$ , 误差定义为:

$$E \stackrel{\text{def}}{=} \sum_{i=1}^n (\mathbf{d}t_i - \mathbf{x}_i + \mathbf{x}_0)^2 = |\mathbf{U}\mathbf{d} - \mathbf{b}|^2 \quad (5.3)$$

由于 $E$ 正比于方程的均方误差, 因此只要 $E$ 达到最小值, 那么点集相对于该直线的波动就最小。换句话说, 也就是该直线最好的模拟了该点集所表示的骨架。由线性代数的方法很容易可以解得 $\mathbf{d} = [(\mathbf{U}^T \mathbf{U})^{-1} \mathbf{U}^T] \mathbf{b}$ 。图5.3展示了由当前节点(蓝色节点)分别向两个点云集合拟合出的两条直线(红色线段), 这两条直线将被作为两个分支的方向。从图中可以看出线性拟合的方法可以很好的估计出树木分枝的方向, 从而准确的恢复出树木的父子结构。

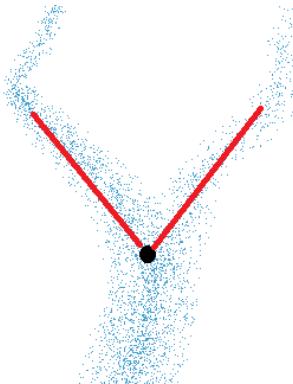


图 5.3 线性拟合计算分支方向

算法5给出了得到邻域信息后进行骨架方向抽取的伪代码，其中*Least Squares Processing*表示运用最小二乘法进行线性拟合。

---

**Algorithm 5** 基于邻域的骨架方向计算
 

---

**Input:** 当前节点体素 $V$

**Input:** 骨架方向数组 $D[1..n]$

**Output:** 当前节点子节点集合 $\$$

```

1: for all 骨架方向 $d \in D$  do
2:    $NewChild \leftarrow LeastSquaresProcessing$ 
3:    $\$.AddChild(NewChild)$ 
4: end for
5: return  $\$$ 

```

---

## 5.4 获取子枝长度和半径

树木骨架的长度和半径对树木模型的真实感有着十分显著的贡献，所以尽可能准确的获得子枝的长度和半径信息能够有助于重建出极具真实感的树木模型。对于树木枝干半径的获取方法有许多，主要分为根据规则生成半径和从树木点云结构中获取半径两种方式。

对于基于规则来生成半径，最简单的方法是对树木半径进行线性地递减，即 $r = cR$ ，其中 $r$ 为子枝半径， $R$ 是父枝半径， $c$ 为一个线性倍数，这个倍数可以固定，也可以进行随机的扰动从而增进多样性。Leonardo da Vinci在经过大量观察后总结出了一种更符合自然规律的树木父子枝直径的关系公式： $D^2 = \sum_{i=1}^n d_i^2$ ，其中 $D$ 为父枝直径， $d_i$ 为第 $i$ 个子枝的直径， $n$ 为子枝的数量。这个公式被广泛地用于树木枝干的半径模拟。

区别于基于规则的半径生成方法，本文为了进一步提升真实感，选择在进行子枝方向抽取的同时，同样进行半径抽取的方法。注意，用该方法的前提是点云分布须均匀化，然而基于图像进行三维重建得到的树木点云会呈现表皮化的现象，这是由于图片上的点都是树木的表皮点，所以在得到三维点云后，是需要进行一些修复工作的，本文用随机点填充的方法对该点云模型进行了实心化的修复。当点云分布满足均匀化时，在对某个骨架进行拟合之后，对于拟合出来的直线，来计算所有参加拟合该直线的点到该直线的平均距离 $D_{avg}$ ，然后就可以计算该骨架的半径 $R = D_{avg} * 2$ 。由于点云分布均匀，所以半径显然就是平均距离的2倍。该算法的伪代码在算法6中给出。

---

**Algorithm 6** 骨架半径抽取
 

---

**Input:** 拟合出的当前子枝所在直线 $L$

**Input:** 当前子枝的点集 $\mathbb{S}$

**Output:** 当前子枝半径 $R$

```

1: 初始化点到直线距离之和 $D_{sum} = 0$ 
2: for all 空间点 $P \in \mathbb{S}$  do
3:   点到直线距离 $D_{sum} += CalculateDistance(P, L)$ 
4: end for
5: 平均距离 $D_{avg} = D_{sum}/\mathbb{S}.Count$ 
6: 子枝半径 $R = D_{avg} * 2$ 
7: return  $R$ 

```

---

图5.4给出了三种半径求解方法的效果对比。5.4(a)给出了线性衰减方法的结果，该方法中子枝半径以父枝半径的线性倍衰减。5.4(b)给出了前文提到的Leonardo da Vinci规则所生成的半径情况。5.4(c)则采用本文中基于线性拟合直线，再计算所有点到该直线平均距离的方法。从三者的效果中可以看出，线性衰减容易出现部分枝条生长不自然的现象，究其原因，还是因为一个单一的绝对的线性系数无法适用于所有的枝条，它对于某些枝条会偏大，对于另外一些枝条会偏小。Leonardo规则虽然给出的是一种父子枝之间的相对关系，从一定程度上解决了线性系数单一绝对而导致的问题，但是它生成的树木枝干会出现过于均与化，而没有捕捉到现实中树木各个局部的特征。本文的方法则由于其基于对所有点的实际恢复坐标进行统计，而更加注重树木的实际局部特征情况，其效果也是三者之中最好的。



图 5.4 三种计算半径方法效果对比

对于骨架长度的估计，基于规则的生成则并不那么具有实践性，因为树木枝干的长度往往并不像半径那样随着父子关系而递减。相反地，它地规则往往要复杂许多，而且并没有统一的规则。基于此考虑，本文并没有对基于规则的长度估计进行实践，而是直接用与半径类似的方法，根据已拟合出的直线，试图从统计的角度对其枝条长度做出合理的估计。一个直观而可行的方法，是将当前节点与所有该方向邻域的点的连线向量投影到拟合出的直线方向向量上，这个投影的平均值的2倍，则可以估计为该方向子枝的长度。本文也给出了子枝长度估计的伪代码，列在了算法7中。

---

**Algorithm 7** 子枝长度抽取

---

**Input:** 拟合出的当前子枝方向向量 $L$ , 父节点 $N$ **Input:** 当前子枝的点集 $\mathbb{S}$ **Output:** 当前子枝长度 $L$ 

- 1: 初始化父节点到邻域点的向量投影长度之和 $D_{sum} = 0$
  - 2: **for all** 空间点 $P \in \mathbb{S}$  **do**
  - 3:     父节点到 $P$ 的向量 $Dir = P.Position - N.Position$
  - 4:      $Dir$ 在 $L$ 上的投影距离 $D_{sum} += CalculateProjectionLength(Dir, L)$
  - 5: **end for**
  - 6: 平均距离 $D_{avg} = D_{sum} / \mathbb{S}.Count$
  - 7: 子枝长度 $L = D_{avg} * 2$
  - 8: **return**  $L$
- 

## 5.5 本章小节

三维体素泛洪是计算机图形学中二维像素泛洪的扩展，它将三维体素按洪水泛滥一般进行空间的扩展。本章利用三维体素泛洪实现了节点对邻域的探索，该方法可以有效的避免枝条间的干扰，并提高邻域探索的效率。然后在该邻域的范围内对方向进行分割，将每个分割方向上的点集根据最小二乘法进行拟合，得出该方向上骨架具体的直线方程。然后根据点集内点到对应直线方程的距离大小，算出该骨架的半径大小，根据点集与当前节点连线在直线方向上的投影长度，来估算出枝干长度。从而得到具有完整信息的骨架结构。

## 第6章 基于枝干合并的轻量化处理

用基于多方向迭代与步长探索得到的三维树木骨架通常是很细致和准确的，尽管它相对于用3DSMAX等建模工具手工建模得到的面片模型已经大大的轻量化了。但是如果应用是用于大规模的树木建模，那么我们有必要根据应用需求进一步进行轻量化处理。

### 6.1 L-System的尝试

#### 6.1.1 L-System简介

L-System是一种并行的重写系统和正规语法，它的结构可以用可以定义为一个3元组：

$$\mathbf{M} = (V, \omega, P)$$

其中：

- **V**(字母表) 表示可以被替代的字符的集合。
- $\omega$ (初始串) 表示L-System的初始状态。
- **P**(规则集合) 表示一系列的衍生规则。

L-System可以根据这三个组成部分的不同而递归地产生形态各异的字符串。由于L-System具有递归生长的特性，因此我们可以用L-System规则来表达一个具有自相似形态或者分形结构的物体，比如本文所研究的对象——树木。

#### 6.1.2 树木模型的参数化L-System规则抽取

球面海龟几何的提出，用参数化的L-System规则描述了树木的结构信息。在球面海龟几何中，节点的空间几何信息用4个量(长度 $l$ 、半径 $r$ 、父子枝夹角 $\theta$ 和水平转角 $\phi$ ) 和4个扩展符号(+、-、&、^)来表示：

- $+(l)$  表示以当前位置为起点，在当前方向上前进 $l$ 单位个长度
- $!(r)$  表示设置当前节点半径为 $r$
- $&(\theta)$  表示设置父子枝夹角为 $\theta$
- $^(\phi)$  设置水平偏角为 $\phi$

在球面海龟几何中，将每个骨架节点生成一条参数化的L-System规则，形如：

$$N(l, r) \rightarrow \&(\theta_0) \wedge (\phi_0)!r + (l)S_0(l*a_0, r*b_0) \dots \&(\theta_n) \wedge (\phi_n)!r + (l)S_n(l*a_n, r*b_n) \quad (6.1)$$

其中N表示当前枝条， $S_0 S_n$ 表示当前枝条的n个子枝条， $a_i b_i$ 分别表示第i个子枝条与当前枝条的长度比和半径比， $\theta_i \phi_i$ 分别表示第i个子枝条与当前枝条的空间夹角和水平偏角。

### 6.1.3 使用L-System进行树木轻量化建模遇到的问题

在用参数化L-System进行树木轻量化建模时，在进行规则归纳时，有个难以克服的问题。考虑将规则6.1中的 $a_0$ 换成 $a'_0$ ，则规则变成一个完全不同的规则。这意味着对于两个分支规则，这两个规则中的子枝的长度，半径，转交，偏角等必须完全相等才能归纳为同一个规则。而对于自然界中形态结构复杂的树木，每个分支规则几乎不可能完全等同于另一个规则。

对上面的问题有一种解决方法就是将参数区间化，将属于同一区间的参数的值视为相同。比如我们可以将父子枝间的转角分为18个区间，每个区间的大小为10度。但是经过分析就可以察觉，这并没有从根本上解决这个问题。假设我们将这4个变量都各自划分为10个区间，那么规则总数最多可以有10000个，而且在这种情况下，两个规律相同的几率也是非常小的。如果我们将分区数量减少，则又有可能将本来差异比较大的规则归纳为一个规则，不符合真实感的要求。

所以，经过分析，这种用参数化L-System进行树木轻量化建模的方法并不适用于从骨架中去抽取规则，而是适用于反向地用其描述的规则去产生一棵树，如台湾学者戴文凯就对单棵树的L-System 规则进行随机扰动而轻量化的建模出了整片森林。

## 6.2 基于枝干合并的树木分级轻量化

用L-System的方法抽取规则所产生的问题，从本质上讲，是由于自然界中的树木形态太复杂和多变。与其从一个本就不规则生长的事物中去抽取规则，还不如直接地在其逻辑结构上进行一系列的轻量化操作。本文提出了分级化地对已抽取的树木骨架中对视觉影响不大的部分进行合并的方法，从而在尽可能

保证模型的视觉效果的基础上，进一步地减小树木模型的体积，使得其能更广泛地应用到WebVR、WebGame等各个领域。

树枝的结构其实只由核心的一些枝干组成，其他的枝干只是对其进行微调。所以在要求进一步轻量化的前提下，本文提出了分别从纵向和横向对树枝进行合并的方法，以去掉一些只是起到微调作用的枝干。这种方法在尽可能保证真实感不过多丢失的前提下对树枝进行简化操作，以适应更广泛的Web应用领域。

### 6.2.1 枝干纵向合并

纵向合并表示从父到子，从根到页进行纵向递归式的合并。若当前节点与其父节点和子节点的夹角小于所设定的阈值，那么则将该节点去掉，并将其子节点连接到其父节点。注意，若该节点的子节点数目不只一个，那么我们不对它进行合并操作，因为将该节点的所有子节点加到该节点的父节点上去有违真实感。图6.1展示了树枝纵向合并过程。图6.1(a)为输入的树枝骨架，并且当前节点为**B**，其父节点为**A**，且只有唯一的子节点**C**。设合并角度阈值为 $\alpha$ ，假设**AB**,**BC**之间的夹角**b**小于合并阈值 $\alpha$ ，那么将**B**剔除，并将**C**作为**A**的子节点。同理，在图6.1(b)中，若夹角**c**小于阈值 $\alpha$ ，那么也将**AC**和**CD**合并。在图6.1(c)中，由于节点**D**有两个孩子，所以不对其进行合并操作。

纵向合并算法的伪代码在算法8中给出。

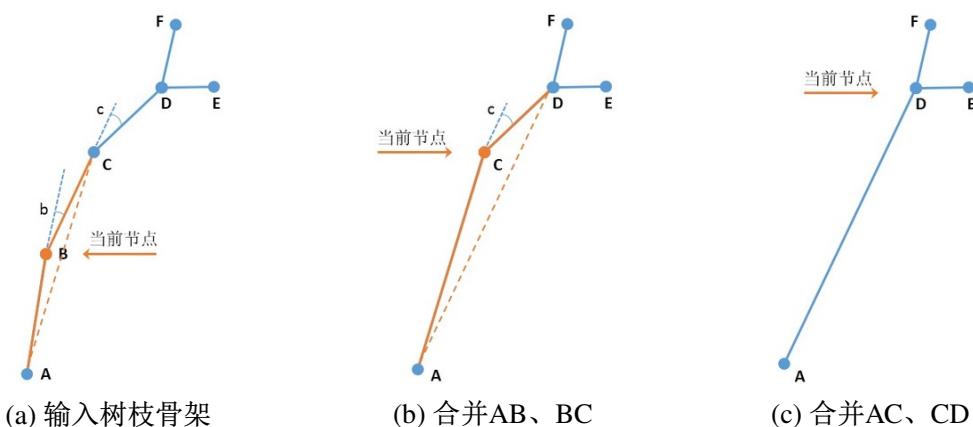


图 6.1 树枝简化过程

---

**Algorithm 8** 纵向合并枝干

---

**Input:** 纵向合并角度 $\alpha$ **Input:** 当前节点 $N$ **Output:** None

```

1: for all 节点 $N' \in N.Children$  do
2:   while  $N'.ChildCount = 1$  do
3:      $\vec{u} \leftarrow N'.Position - N.Position$ 
4:      $\vec{v} \leftarrow N''.Position - N'.Position$ 
5:      $\gamma \leftarrow \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|}\right)$ 
6:     if  $\gamma < \alpha$  then
7:        $N.child \leftarrow N.AddChild(N'')$ 
8:        $N.child \leftarrow N.DeleteChild(N')$ 
9:     end if
10:     $N' \leftarrow N'.FirstChild$ 
11:  end while
12: end for
13: if  $N.ChildCount > 1$  then
14:   for all 节点 $N' \in N.Children$  do
15:     以 $N'$ 为当前节点递归调用该函数
16:   end for
17: end if

```

---

### 6.2.2 枝干横向合并

横向合并指的是对非常靠近的叶子节点进行合并。之所以只对叶子节点进行合并，是因为非叶子节点下面都有若干棵子树，若对它们进行合并，必须对它们下面的子树也进行合并。而合并子树显然就使得真实感下降很大，因为这不只是局部微调，而是若干子树的变动。对于横向合并，不再是使用角度来衡量两个子枝的靠近程度，而是使用子节点间的欧式距离来表示，因为合并两个角度相差小但是长度相差大的子枝也会导致真实感的大幅下降。横向合并的伪代码在算法9中给出。

---

**Algorithm 9** 横向合并枝干

---

**Input:** 初始化横向合并距离阈值 $\mu$ **Input:** 设定当前节点 $N$ **Output:** None

```

1: for all 节点对 $P \in N.Children$  do
2:    $N_1 \leftarrow P.FirstNode$ 
3:    $N_2 \leftarrow P.SecondNode$ 
4:   if  $N_1.ChildCount = 0 \wedge N_2.ChildCount = 0$  then
5:      $\tilde{\mathbf{u}} \leftarrow N_1.Position$ 
6:      $\tilde{\mathbf{v}} \leftarrow N_2.Position$ 
7:      $\gamma \leftarrow |\tilde{\mathbf{u}} - \tilde{\mathbf{v}}|$ 
8:     if  $\gamma < \mu$  then
9:       New Node  $N'$ 
10:       $N'.Position \leftarrow (N_1.Position + N_2.Position)/2$ 
11:       $N'.Radius \leftarrow \max(N_1.Radius, N_2.Radius)$ 
12:       $N.child \leftarrow N.DeleteChild(N_1)$ 
13:       $N.child \leftarrow N.DeleteChild(N_2)$ 
14:       $N.child \leftarrow N.AddChild(N')$ 
15:      退出循环并以当前节点 $N$ 重新调用该函数
16:    end if
17:  end if
18: end for
19: for all 节点 $P \in N.Children$  do
20:   以 $P$ 为当前节点递归调用该函数
21: end for

```

---

### 6.2.3 联合使用横纵向合并

纵向合并和横向合并单独使用时都具有很大的局限性，因为纵向合并只能对具有单个孩子并且没有兄弟的节点进行纵向递归地调用，而横向合并又只能对叶子节点进行兄弟级别的合并。但是将两种合并方法联合使用，将可以从整体上对树木进行微调操作，图6.2对这一想法进行了演示。6.2(a)中经过纵向的AC,CD合并得到6.2(b)。6.2(b)中由于D有两个子节点，无法进行纵向合

并，所以考虑进行横向合并DE,DF，并得到6.2(c)。最后进行一次纵向合并得到6.2(d)。

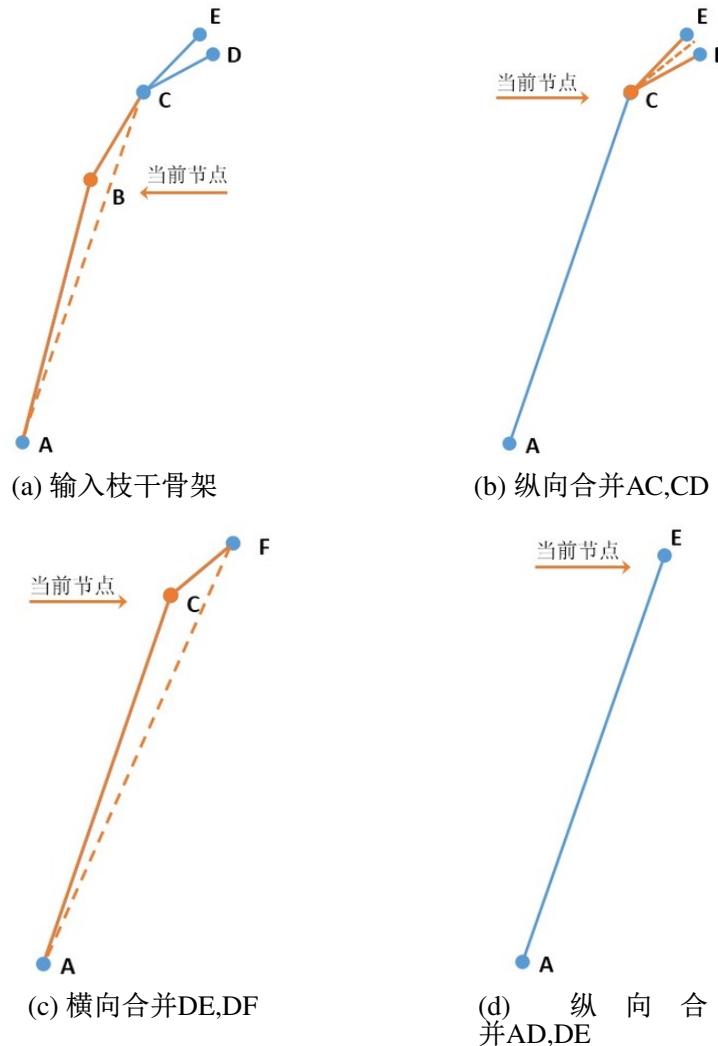


图 6.2 联合使用纵向和横向合并

### 6.3 本章小节

在得到骨架结构后，为了迎合轻量化的应用，本文对其进行轻量化操作。本章首先对传统的轻量化方法L-System进行了尝试，运用参数化的L-System规则进行规则抽取，但是由于现实中树木的复杂性与不规则性，抽取出来的规则太多，以至于违背了轻量化的原则。于是本文提出了基于枝干合并的轻量化方法，分别从纵向和横向对树木枝干进行合并，并对这两种方法进行有机的组合使用，以达到轻量化的目标。

## 第7章 建模还原度

对于一个通过建模获得的树木模型，如果没有一个客观的量化评价指标，就无法从客观的角度反馈树木模型的还原度和各个步骤算法的可行性。对于本文的基于图像序列的树木建模方法，建模的输入是在自然环境下拍摄的树木图片序列，输出是三维的骨架模型。因此，判断三维模型和投影照片的相似程度是评价建模质量的核心。然而，大多数基于图像的树木建模论文<sup>[8-11,19]</sup>只给出了输入图片和建模结果在少量角度的渲染效果，试图让观察者从肉眼观察其相似度。但是这种方法是主观的，因观察者的不同可能会有不同的评价结果，这显然不是一个好的评价方法。

为了客观、量化地评价基于图像序列的建模质量，本文提出了一套完整的评价方法。然而，仅仅凭借照片无法完全表达出其所在环境的信息，比如环境光照，因为遮挡而产生的阴影信息等，因此本文的评价方法将不针对模型的纹理和颜色信息，仅仅对模型的几何信息和照片中的几何信息的匹配程度进行量化分析。

设树木模型 $M$ 由 $n$ 张从不同角度拍摄的同一棵树的图片序列 $I_1 \sim I_n$ ，经过基于图像的三维重建，骨架抽取的方法进行建模所获得。那么模型 $M$ 的建模还原度 $\mathbb{Q}$ 定义如下：

**定义 7.1：**

$$\mathbb{Q} = \mathbf{I} \cdot \mathbf{R}_{3d} \cdot \mathbf{R}_s$$

建模还原度 $\mathbb{Q}$ 的取值范围为 $[0, 1]$ ，0表示没有还原出任何树木几何信息，1表示准确还原出整棵树木的几何信息。

本文将建模还原度 $\mathbb{Q}$ 考虑由3个部分组成，为此也引入了三个新的概念：图片序列信息量 $\mathbf{I}$ ，三维重建还原度 $\mathbf{R}_{3d}$ ，以及骨架抽取还原度 $\mathbf{R}_s$ 。这三个分量的取值范围都为 $[0, 1]$ ，它们的乘积即为总的建模还原度 $\mathbb{Q}$ 。后续小节会详述这三个分量。

### 7.1 图像序列信息量

当实地对树木进行多角度拍摄时，拍摄者将基于不同的水平角度对树木进行全方位的拍摄，以便将整棵树的信息尽可能多的携带进图像序列中。然而，

从客观上来看，怎么样的图片序列才更加完整的表达了整棵树的几何特征？为了从客观和量化角度给出树木图像序列所携带的树木信息的多少，本文引入了图像序列信息量的概念。

那么，到底怎样的图片序列携带的信息量更大呢？从拍摄过程分析，如果想要得到一棵完整的树木信息，那么需要绕着一棵树一圈进行密集地拍摄。这里的一圈，用数学化的表示，就是 $360^\circ$ ，如果只是绕半圈进行拍摄，那么所得到的图片序列表达的树木信息必定是不完整的，所以角度对信息量有着很大的贡献。另一方面，如果每隔 $60^\circ$ 进行一次拍摄，和每隔 $30^\circ$ 进行一次拍摄，在它们都绕圈拍摄的前提下，后者的图片序列所含信息量必然更大。再进一步思考，如果我隔 $360^\circ, 180^\circ, 90^\circ, \dots, 1^\circ$ 进行拍摄呢？那么后一次拍摄所得的图片序列相比前一次的图片序列信息量的增长是相同的吗？答案是否定的，因为当图片很少时，三维重建的结果也不好，这时增加图片数量是能够很大程度上改观三维重建的重量的，因此此时的信息量增长速度快。但是在拍摄已经比较密集的情况下，后一次拍摄所增加的信息量必定只是一些细节的信息，所以，信息量增长的速率应该变小。并且一个信息量大的图片序列应该满足以下三个要求：

- **图片数量多：**图片数量多也就意味着拍摄角度多，因为一张图片代表着一个角度。
- **角度跨度大：**跨度大指需要对树木进行全方位的拍摄。
- **角度分布均匀：**若图片只是密集的集中在一个角度区间，就算图片再多，也无法完整地表达整棵树的信息，所以若在角度多和跨度大的情况下还满足分布均匀，那么就能很完整地携带树木的信息。

由于从平面的2D图像很难得到其空间角度拍摄情况，因此在这里我们简化其定义，将关注点放在图片数量上来，对于图片跨度和角度的均匀分布，我们默认拍摄者在拍摄过程中采用均匀的角度偏差来进行 $360^\circ$ 的拍摄。

根据以上的分析，本文给出了图像序列信息量的数学定义如下：

**定义 7.2：**

$$I = 1 - \left[ \frac{a}{b} \right]^n$$

其中，图片序列信息量 $I$ 的取值范围为 $[0, 1]$ 。当 $I = 0$ 时表示图片序列并不包含树木信息，当 $I = 1$ 时表示图片序列能完全表达空间树木的几何信息。 $a, b$ 都是正数且 $a < b$ ，具体数值需要对不同树木进行实验之后才能得到。尽管 $a$ 和 $b$ 因树

木特点不同而不同，但是它始终满足前文提出的信息量增长速度的特点，即先快后慢。

## 7.2 三维重建还原度

对于一个给定的图片序列，所用三维重建方法所得到的点云模型的与实际的树木在几何形状上的相似度如何，由三维重建还原度 $\mathbf{R}_{3d}$ 来定义。注意，实际树木的几何信息被记录在输入的图像序列中，所以想要计算点云模型和实际树木的相似度，就需要对点云模型和图片序列进行比较。然而对于三维的点云信息和二维的图片信息，无法进行直接地比较。一个比较直观的想法，是对三维的点云进行投影，投影的角度由三维重建过程中的照相机几何标定步骤给出。

由于不考虑模型纹理和颜色信息，在空间点被投影到平面以后，只关注其是否在对应角度图片的树木轮廓内。所以对输入的树木图片序列，需要先获得其轮廓图，并将其转化为二值图像。树木上的点值为1，而树木外的点值为0。对于每一个点云模型中的点，按对应角度投影，获得其在对应图片上的坐标值，并且在其二值图像上确定其值，若为1，则表明匹配成功，否则匹配失败。最后统计出匹配成功的总的比例，作为三维重建的还原度。

根据以上分析，本文给出三维重建还原度的数学定义式：

**定义 7.3：**

$$\mathbf{R}_{3d} = \frac{1}{n} \sum_{i=1}^n \frac{P_i}{P_i + O_i}$$

上式中的 $n$ 表示图像的数量， $P_i$ 表示点云模型投影到第 $i$ 张图片上在树木轮廓中的点的数量， $O_i$ 表示点云模型投影到第 $i$ 张图像上在树木轮廓外的点的数量，因此 $P_i + O_i$ 自然就表示点云模型中点的总数量。 $\frac{P_i}{P_i + O_i}$ 表示点云投影到第 $i$ 张图片上的击中率。最后对每张图像的击中率求平均，作为总的三维重建的还原度。其值区间为[0, 1]。

## 7.3 骨架抽取还原度

骨架抽取是基于三维点云模型进行的，因此计算骨架抽取的还原度的输入是重建出的点云模型和抽取出的骨架模型。由于点云模型是三维的点的集合，而抽取出的骨架模型却是一个记录着树形结构的逻辑信息，它们无法进行直接

的比较。本文采取的做法是将骨架的树形逻辑信息用圆台和球进行堆叠从而将其转化为三维的表示。

具体的做法是对骨架中的每个节点，根据其半径构造出一个球体。然后对于每个父子关系，用一个圆台来表示其枝干，圆台的底半径等于父节点的半径，圆台的顶半径等于子节点的半径。然后对于每个点云模型中的点，用数学公式判断其是否存在于骨架的三维表示中的球体或圆台中，如果存在，则表示匹配成功，否则表示匹配失败。最后用成功点数与总点数的比值来表示骨架抽取的还原度。定义如下：

**定义 7.4：**

$$\mathbf{R}_s = \frac{S}{N}$$

其中  $S$  表示匹配成功的点数，而  $N$  表示点云模型的总点数。 $\mathbf{R}_s$  的值区间为  $[0, 1]$ 。

注意，若用经过枝干合并轻量化处理的骨架进行骨架抽取还原度计算，其值必定会比直接从点云中抽取出来的模型要小，因为模型经过简化后，与原点云模型的匹配度也必将降低。本文的目标只是尽可能在还原度降低不多的情况下，对骨架进行尽可能多的轻量化。

## 7.4 建模还原度计算

将图片序列信息量  $\mathbf{I}$ 、三维重建还原度  $\mathbf{R}_{3d}$  和骨架抽取还原度  $\mathbf{R}_s$  代入建模还原度  $\mathbf{Q}$  的定义式中，可以得到建模还原度的计算式：

$$\mathbf{Q} = (1 - [\frac{a}{b}]^n) \cdot \frac{1}{n} \sum_{i=1}^n \frac{P_i}{P_i + O_i} \cdot \frac{S}{N} \quad (7.1)$$

## 7.5 本章小节

本章提出了基于图像树木轻量化建模的质量评价方法。首先提出了建模还原度的概念，它包含了三个子项：图像序列信息量、三维重建还原度以及骨架抽取还原度。它们分别代表了图像序列对真实树木的信息携带量、点云模型和图像序列的匹配度、骨架模型与点云模型的匹配度。本文对这三个子项的由来和计算方法都进行了阐述，并将它们融合给出了建模还原度的计算式。

## 第8章 基于用户交互的模型改善

通过自动化算法生成的模型难免含有算法设计者的主观想法在其中，比如树木何时应该分支，何时应该合并等等。并且算法在处理实际的模型时，通常带有一定的二义性。未免过于主观地决定了模型的最终成型，本文提倡在自动化生成模型以后，应该将模型的建立延迟到应用程序，或者使用该模型的最终用户。这样可以使得本文所建立的一系列方法适用于更广的情形。将自动化算法与人工交互联合使用，可以大大地提高最终模型与具体需求之间的耦合度。

本文根据此需求，开发完成了一个树木模型用户交互平台，可以用于对本文的基于图像所建立的树木模型进行进一步地编辑和完善。该平台集成了几种不同轻量化级别的模型表示，模型的编辑，算法的演示等功能，大大的提高了算法验证的直观度和用户与应用级模型改善的方便性。该平台的具体功能见用例图8.1。

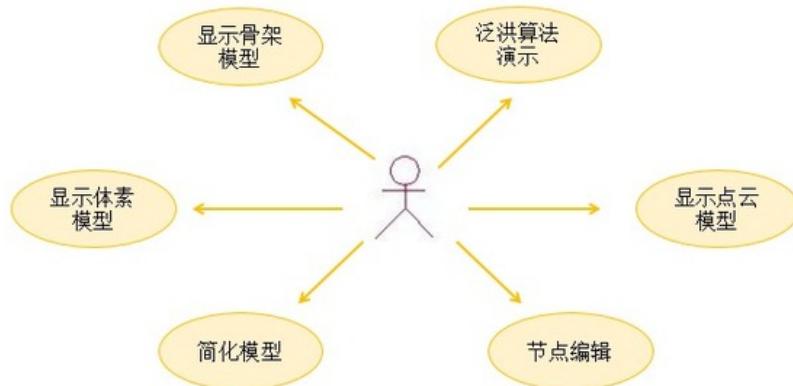


图 8.1 用户交互平台用例图

对于每个用例，本文用表格和图片对其进行进一步说明，由于本文并不将重点放在软件开发上，所以在本文中未给出具体的系统分析和设计。本文试图对该用户交互平台的功能进行阐述，从而在功能性上对自动化算法进行人工加强和改善。

## 8.1 加载树木点云文件

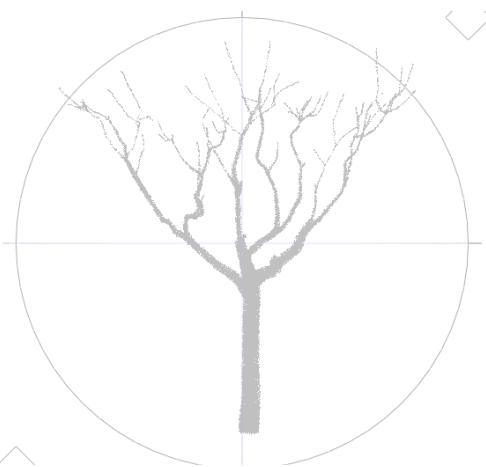


图 8.2 用例1图示：用开源软件MeshLab打开的树木点云文件

用例名称:	加载树木点云文件
用例标志号:	1
参与者:	用户
简要说明:	用户可以按键'1'来加载存储为".ply"开源格式的名为"Tree.ply"的点云文件，该后缀名的点云模型可以用开源软件Meshlab来打开观察，如图8.2
前置条件:	树木的模型文件存在于./Models/文件夹下
基本事件流:	<ol style="list-style-type: none"><li>1. 用户按下'1'键</li><li>2. 系统去./Models/文件夹下搜索名为"Tree.ply"的点云文件</li><li>3. 将点云文件加载入内存</li><li>4. 用例终止</li></ol>
异常事件流:	<ol style="list-style-type: none"><li>1. 若点云文件不存在，则不会加载入内存</li></ol>
后置条件:	无

## 8.2 显示骨架模型

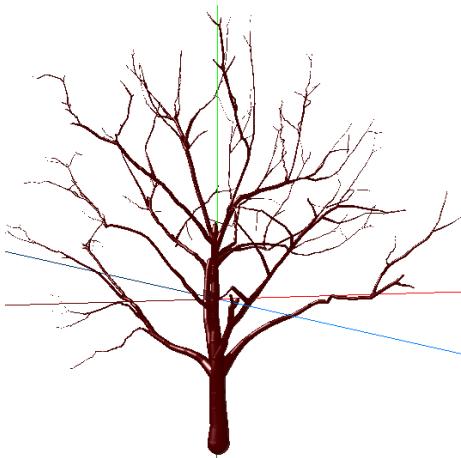


图 8.3 用例图示：显示骨架模型

用例名称:	显示骨架模型
用例标志号:	2
参与者:	用户
简要说明:	用户可以通过选择显示骨架模型来观察树木的骨架，具体效果见图8.3
前置条件:	树木的模型文件已经被加载进入软件平台
基本事件流:	<ol style="list-style-type: none"><li>1. 用户按下'm'键将显示模式切换到骨架模式</li><li>2. 将树木骨架模型用冯氏光照模型渲染至视口</li><li>3. 用例终止</li></ol>
异常事件流:	<ol style="list-style-type: none"><li>1. 若模型文件未加载，则不会显示对应模型骨架</li></ol>
后置条件:	无

### 8.3 显示点云模型

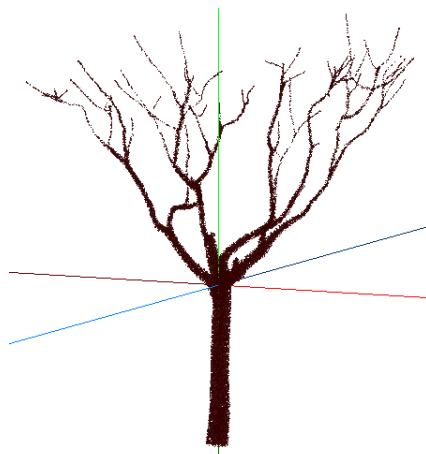


图 8.4 用例图示：显示点云模型

用例名称:	显示点云模型
用例标志号:	3
参与者:	用户
简要说明:	用户可以通过选择显示点云模型来观察树木的点云表示，具体效果见图8.4
前置条件:	树木的模型文件已经被加载进入软件平台
基本事件流:	<ol style="list-style-type: none"><li>1. 用户按下'm'键切换显示模式到点云模式</li><li>2. 将点云模型用平滑光照模型渲染至视口</li><li>3. 用例终止</li></ol>
异常事件流:	<ol style="list-style-type: none"><li>1. 若模型文件未加载，则不会显示对应点云模型</li></ol>
后置条件:	无

## 8.4 显示体素模型

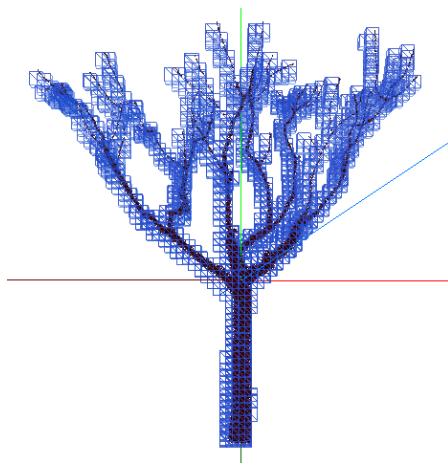


图 8.5 用例图示：显示体素模型

用例名称:	显示体素模型
用例标志号:	4
参与者:	用户
简要说明:	用户可以通过选择显示体素模型来观察树木的体素表示，具体效果见图8.5
前置条件:	树木的模型已经被加载进入软件平台
基本事件流:	<ol style="list-style-type: none"><li>1. 用户按下'v'键</li><li>2. 将体素模型用冯氏光照模型渲染至视口</li><li>3. 用例终止</li></ol>
异常事件流:	<ol style="list-style-type: none"><li>1. 若模型文件不存在，则不会显示对应点云模型</li></ol>
后置条件:	无

## 8.5 节点编辑

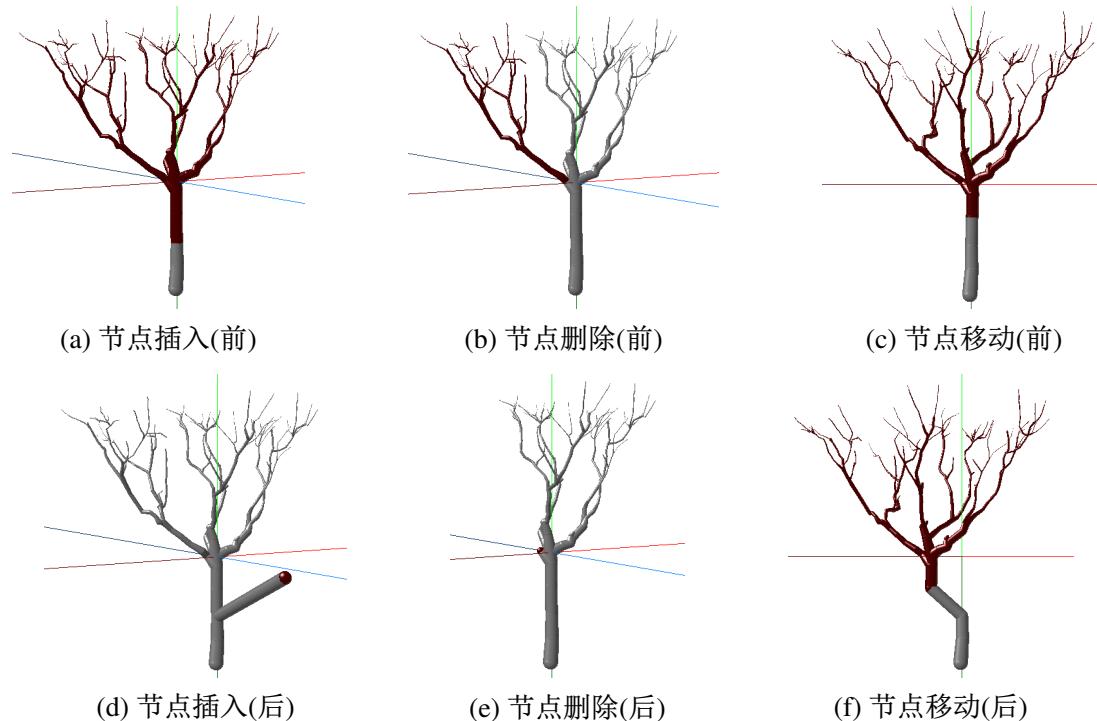


图 8.6 用例图示：节点编辑

用例名称:	节点编辑
用例标志号:	5
参与者:	用户
简要说明:	用户可以通过选中特定的节点，对其进行丰富的编辑操作，具体效果见图8.6
前置条件:	树木的显示模式为骨架模型
基本事件流:	<ol style="list-style-type: none"> <li>1. 用户按下'g'键</li> <li>2. 树木骨架从当前节点按用户视角右方向生长出一个子节点</li> <li>3. 调整当前节点为新增的子节点</li> </ol>
可选事件流1:	<ol style="list-style-type: none"> <li>1. 用户按下'p'键</li> <li>2. 树木删除当前节点以下的子树</li> </ol>
可选事件流2:	<ol style="list-style-type: none"> <li>1. 用户按下鼠标左键拖动</li> <li>2. 当前节点及其以下的子树在当前平面内平移</li> </ol>
可选事件流3:	<ol style="list-style-type: none"> <li>1. 用户按下'n','f'键</li> <li>2. 当前节点及其以下的子树在z平面内外平移</li> </ol>
后置条件:	无

## 8.6 简化模型

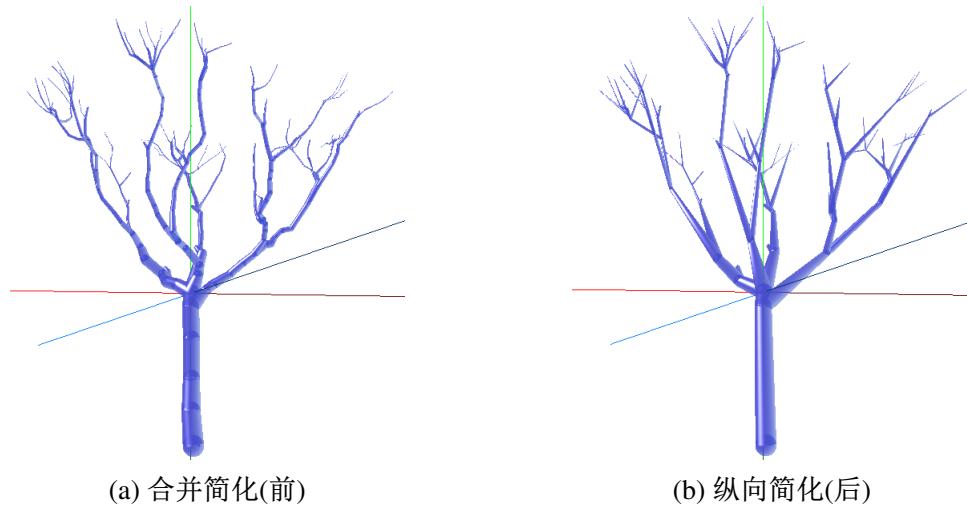


图 8.7 用例图示：简化模型

用例名称:	简化模型
用例标志号:	6
参与者:	用户
简要说明:	用户可以通过选择某棵子树，对其进行如第6章中的合并操作，以简化模型。具体效果见图8.7
前置条件:	树木的显示模式为骨架模型
基本事件流:	<ol style="list-style-type: none"><li>1. 用户按下's'键</li><li>2. 对当前节点及其表示的子树进行基于合并的简化</li><li>3. 用例终止</li></ol>
后置条件:	无

## 8.7 泛洪算法演示

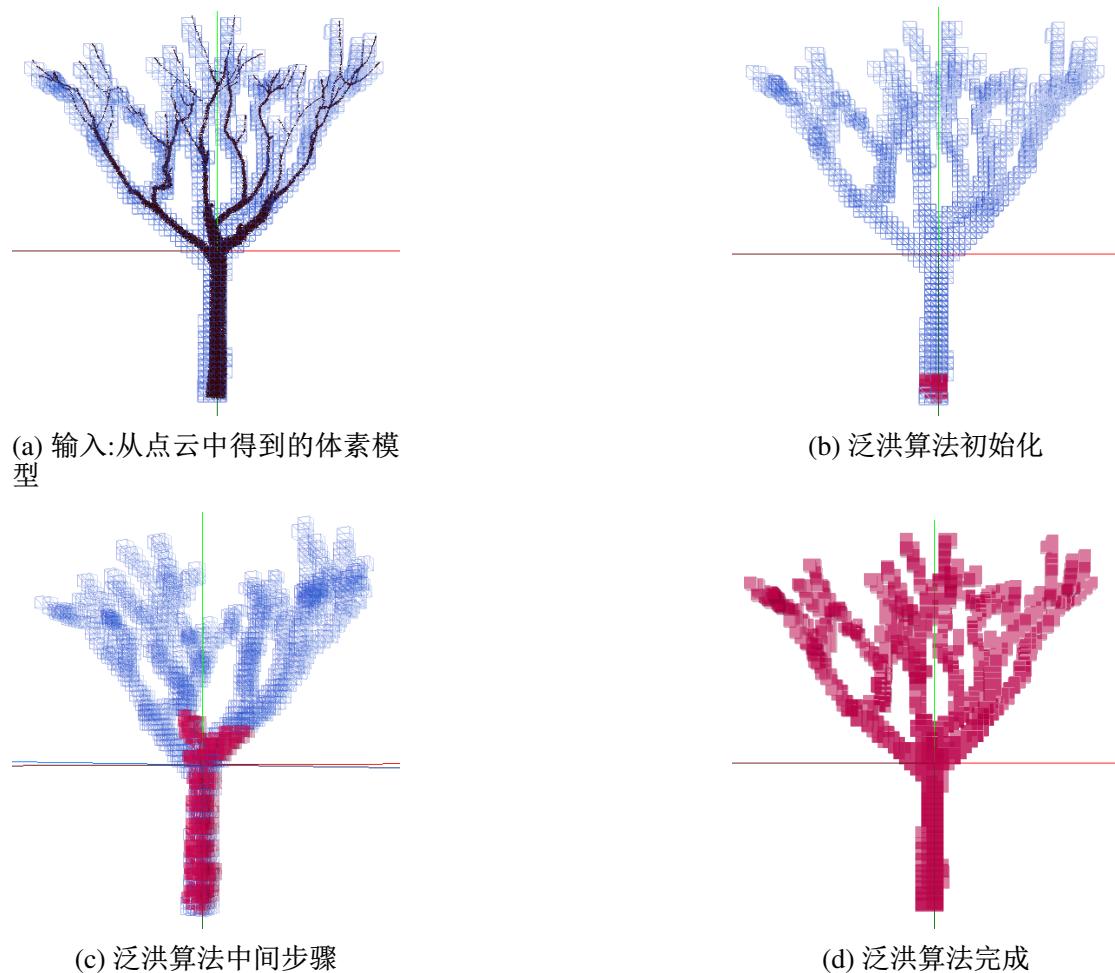


图 8.8 用例图示：泛洪算法演示

用例名称:	泛洪算法演示
用例标志号:	7
参与者:	用户
简要说明:	用户可以根据输入的点云，进行泛洪算法的可视化单步观察。具体效果见图8.8
前置条件:	树木的点云数据已经被加载进入程序，并且打开了体素模型模式
基本事件流:	<ol style="list-style-type: none"> <li>1. 用户按下'e'键</li> <li>2. 对当前节点进行泛洪并找到其子节点</li> <li>3. 选中其子节点，并重复1和2中的步骤</li> <li>4. 当达到叶子节点时，用例结束</li> </ol>
后置条件:	无

## 第9章 实验过程与分析

### 9.1 实验环境

对于前面提出的基于图像的树木轻量化建模方法，本文做了大量实验验证了其可行性。本文使用的拍摄工具是高分辨率安卓手机索尼LT26ii，该手机最高分辨率能达到4000x3000，已经足够实验的需求。拍摄地点为同济大学嘉定校区以及某住宅小区。本文实验所使用计算机操作系统平台为XUbuntu 12.04，CPU为双核Intel(R) Core(TM) i5-3230M @ 2.60GHz。显卡为NVidia GeForce GT750M。所有实验程序均使用C/C++语言完成，使用g++编译器进行编译，同时使用OpenGL version 4.3图形硬件接口完成了第八章提及的可视化平台。

### 9.2 实验结果与分析

本文以三棵树的建模结果作为展示和分析的依据，其中第一棵树拍摄帧数为11帧，见图9.1。第二棵树的拍摄帧数为12帧，见图9.2。第三棵树的拍摄帧数为20帧，见图9.3。

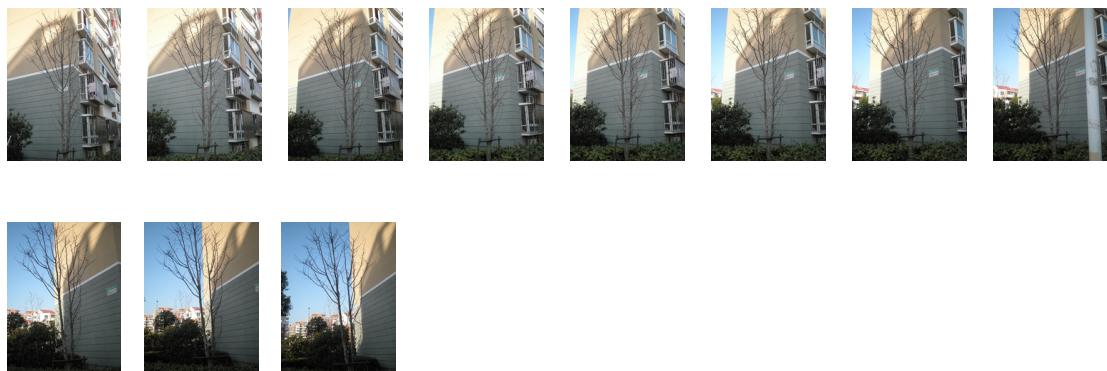


图 9.1 树木1图像序列，帧数为11帧，拍摄地点为某住宅小区

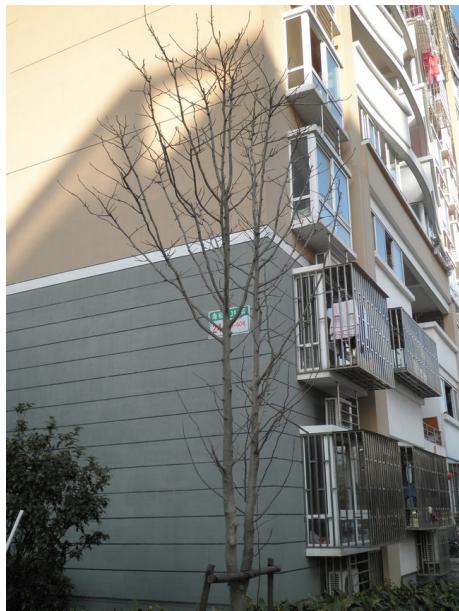


图 9.2 树木2图像序列，帧数为12帧，拍摄地点为同济大学嘉定校区

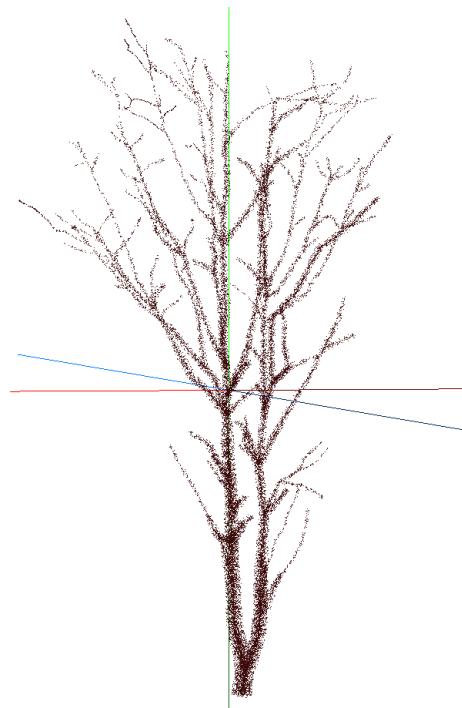


图 9.3 树木3图像序列，帧数为20帧，拍摄地点为同济大学嘉定校区

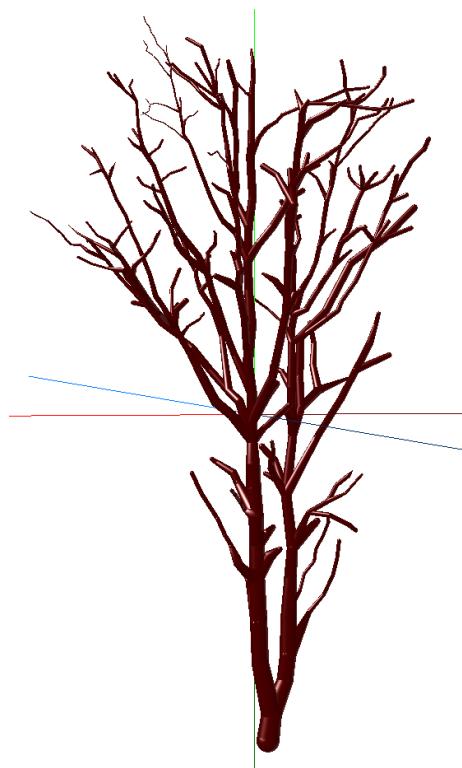
本文还给出了每个图片序列三维重建出的点云模型、从点云模型中直接抽取的骨架模型以及经过枝干合并方法轻量化得到的骨架模型。为了从直观上比较重建模型和图片序列的相似度，本文从正面和侧面对所得模型进行投影，以方便从二维的视角判断其相似程度。图9.4和图9.5分别给出了树木样本1在正面投影和侧面投影的比对情况。图9.6和图9.7分别给出了树木样本2在正面投影和侧面投影的比对情况。图9.8和图9.9分别给出了树木样本3在正面投影和侧面投影的比对情况。



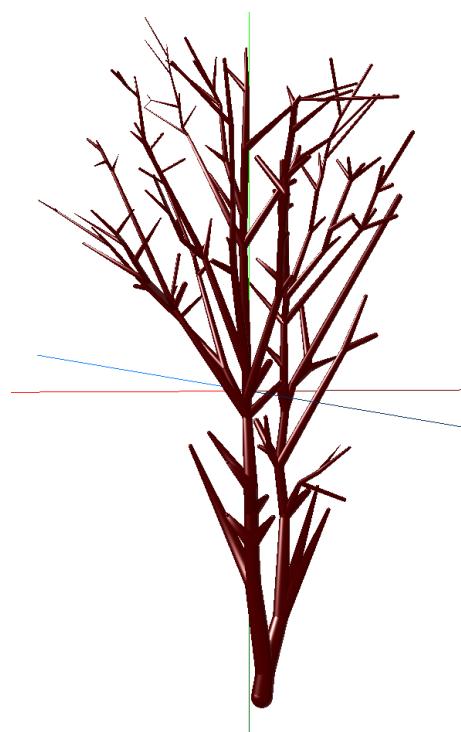
(a) 树木正面图像



(b) 树木点云模型



(c) 树木骨架模型



(d) 树木轻量化骨架模型

图 9.4 树木样本1正面投影比对



图 9.5 树木样本1侧面投影比对

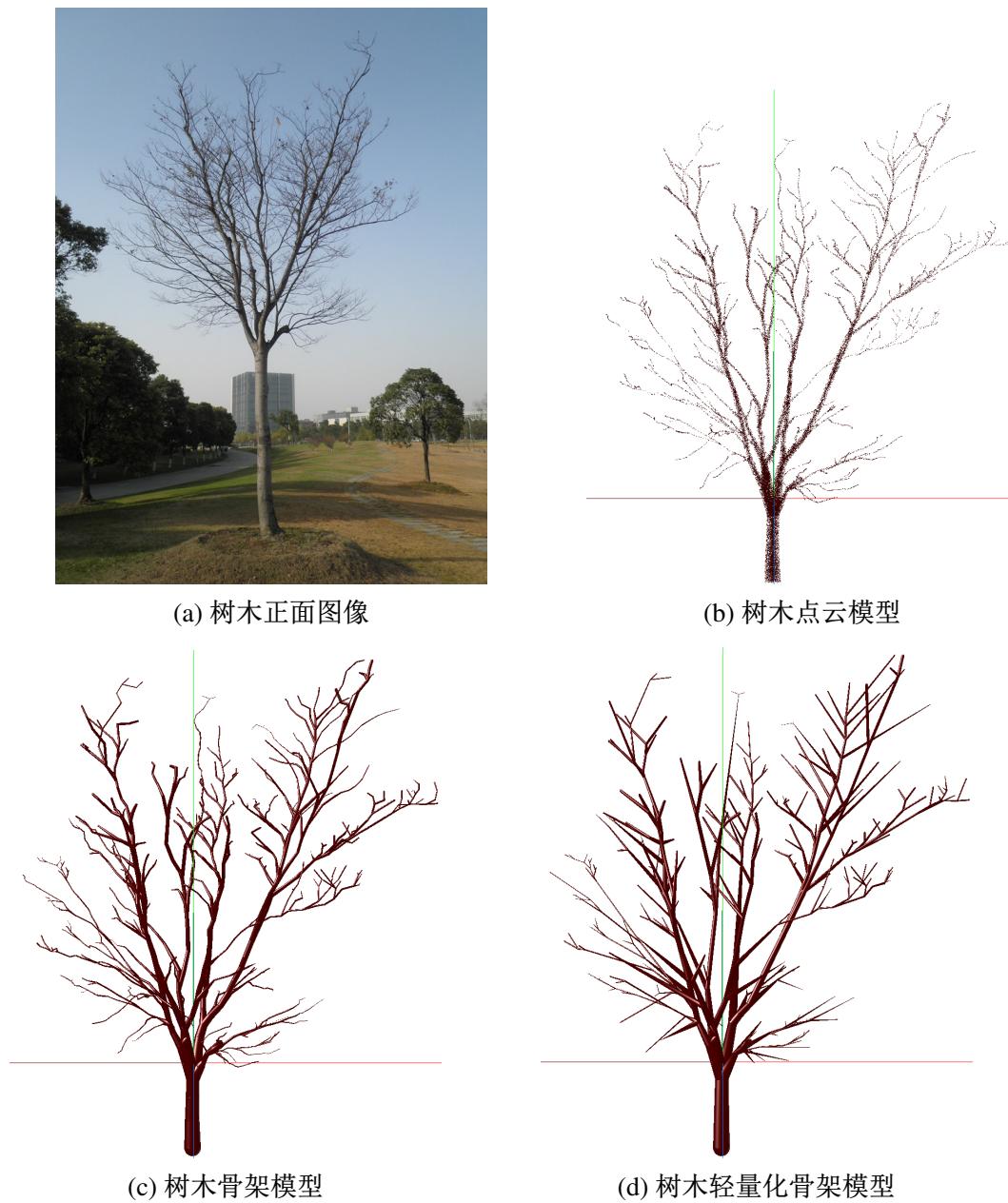


图 9.6 树木样本2正面投影比对

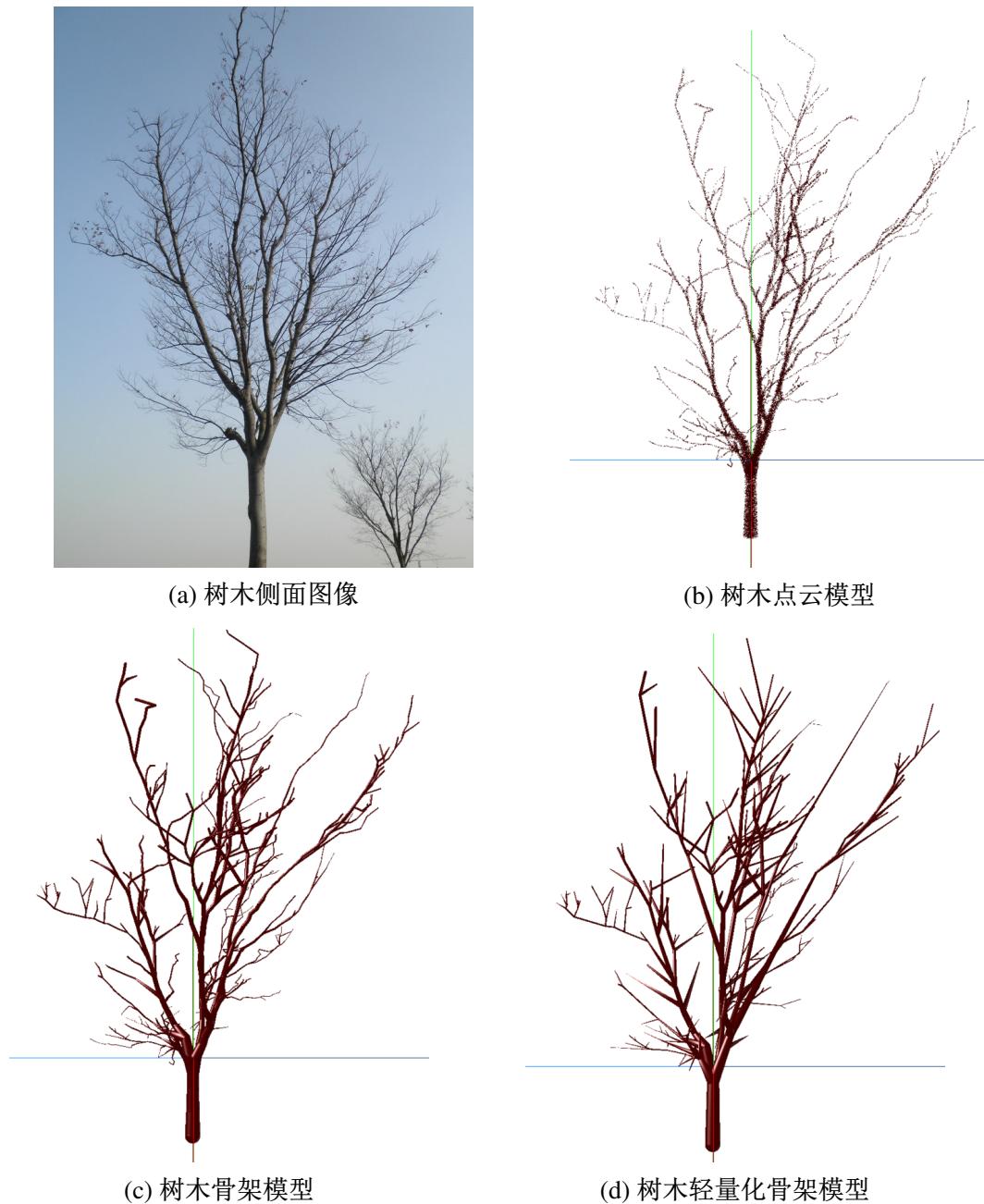


图 9.7 树木样本2侧面投影比对



图 9.8 树木样本3正面投影比对

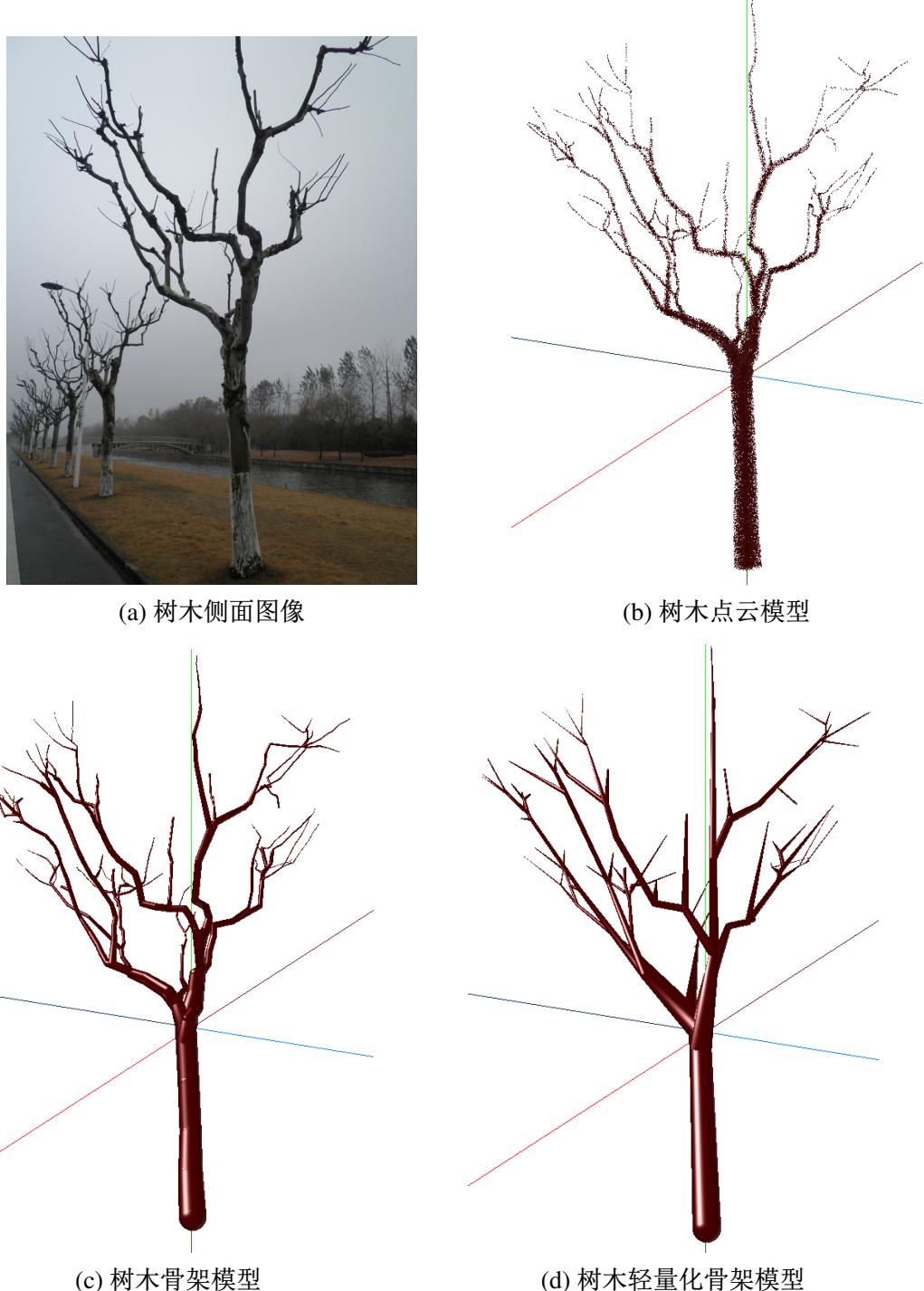


图 9.9 树木样本3侧面投影比对

可以看出，无论从正面还是侧面，通过本文轻量化建模方法所得到的树木模型都具有很高的还原性。由于树木1和树木的细小枝干比较多，所以对于其细节部分的还原度还有待提升。但是类似树木3这种不含太多细小枝干的树木，本文可以给出很高的还原度。从另一方面来看，由于本文的方法是轻量化建模，对于细小的枝干的舍弃也是无法避免的，所以从总体来看本文的方法对于真实树木的轻量化建模效果还是十分客观的。

表9.1分别对图片序列信息量、三维重建还原度、骨架抽取还原度以及总的还原度给出了量化的实验计算结果。注意，图片序列信息量的计算，本文根据实验所得的最佳 $a/b = 0.8$ 的结果来进行计算。从表中可以看出，树木样本3的还原度最高，而树木样本2的还原度最低。分别比较它们的三维重建还原度和骨架抽取还原度可以看出，图片数量的越多，样本结构越简单，则三维还原度的还原度就越高。而对于树木结构复杂的树木，骨架抽取是一个难点，因为树木分支结构的复杂性容易引起骨架抽取的二义性从而导致抽取不准确，所以对于样本3这样结构简单的树木来说，骨架抽取还原度最高，而对于结构最复杂，细枝最多的样本2来说，骨架抽取还原度最低。

表 9.1 树木样本还原度统计

树木样本	图片序列信息量	三维重建还原度	骨架抽取还原度	总还原度
1	0.988	0.892	0.871	0.767
2	0.931	0.849	0.852	0.673
3	0.988	0.935	0.953	0.881

表9.2从轻量化的角度，对比了三个树木样本从三维重建得到的点云数据文件的体积、骨架抽取后得到的文件体积和经过轻量化以后得到的文件体积。从表中可以看出，三个样本从点云数据到骨架数据体积都大大的减小了，对于以KB为单位的骨架体积，已经可以普及到一般的Web应用来。对于后一步轻量化的步骤，主要是为了满足更高的应用需求，可以看出，对于骨架结构中细枝较多的样本1和样本2，其轻量化所减小的模型体积比例要大于结构简单的样本3，这是因为模型的轻量化主要是建立在对树木结构进行简化，而本来就比较简单的结构，这种轻量化的程度就会减弱。

表 9.2 树木建模各阶段文件体积对比

树木样本	三维重建点云体积	原骨架体积	轻量化骨架体积
1	2.5M	50.3K	8.2K
2	4.4M	103.1K	15.1K
3	2.2M	31.5K	7.3K

### 9.3 本章小节

本章首先交代了本文所进行实验的软硬件平台等实验环境。然后给出了3个样本的实验结果，从模型正面和侧面进行投影并从直观上比较了重建模型与输入图像序列的相似度。然后进一步运用前一章给出的建模还原度的计算式，对三个样本进行了量化的计算，并分析了影响建模还原度的一些因素。最后给出了3个样本重建模型的轻量化数据，并基于该分析了影响树木模型轻量化的一些因素。从最后的实验结果，我们可以看出，本文提出的基于图像的轻量化建模方法具有可行性和十分客观的重建效果。

## 致 谢

衷心感谢我的导师贾金原教授。在这一年的毕业设计过程中，贾老师严谨的治学作风、渊博的学术造诣以及悉心的指导给了我莫大的帮助和激励。不仅使我学到了如何高效率地工作，也让我学到了在治学方面所应该持有的执着又平淡的态度。使我从一个对计算机图形方面一无所知的门外汉，成为了深入其中并乐此不疲的钻研者。在论文完成之际，对贾老师表示最衷心的谢意。

感谢同济大学软件学院在这本科四年中对我的培养，让我学会了做人，也让我对软件开发产生了浓厚的兴趣。感谢同济大学，你终将是我引以为傲的母校。在此希望学校和学院都越来越好。

感谢实验室全体同学对我的帮助和支持。本课题承蒙国家自然科学基金资助，特此致谢。

感谢LATEX的存在，虽然它没有所见即所得的直观，但却为我排版省下了不少时间，也使我的论文格式漂亮了很多。

## 参考文献

- [1] Lindenmayer A. Mathematical models for cellular interaction in development: Parts I and II. *Journal of Theoretical Biology*, 1968, 18.
- [2] Prusinkiewicz P, Lindenmayer A. The algorithmic beauty of plants. New York, NY, USA: Springer-Verlag New York, Inc., 1990.
- [3] Prusinkiewicz P, Lindenmayer A, Hanan J. Development models of herbaceous plants for computer imagery purposes. *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM, 1988. 141–150.
- [4] Anastacio F, Prusinkiewicz P, Sousa M C. Sketch-Based Interfaces and Modeling (SBIM): Sketch-based parameterization of L-systems using illustration-inspired construction lines and depth modulation. *Comput. Graph.*, 2009, 33(4): 440–451.
- [5] Reffye P, Edelin C, Françon J, et al. Plant models faithful to botanical structure and development. *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: ACM, 1988. 151–158.
- [6] Reche-Martinez A, Martin I, Drettakis G. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Trans. Graph.*, 2004, 23(3): 720–727.
- [7] Neubert B, Franken T, Deussen O. Approximate image-based tree-modeling using particle flows. *ACM Trans. Graph.*, 2007, 26(3).
- [8] Quan L, Tan P, Zeng G, et al. Image-based plant modeling. *ACM SIGGRAPH 2006 Papers*, New York, NY, USA: ACM, 2006. 599–604.
- [9] Tan P, Zeng G, Wang J, et al. Image-based tree modeling. *ACM SIGGRAPH 2007 papers*, New York, NY, USA: ACM, 2007.
- [10] Tan P, Fang T, Xiao J, et al. Single image tree modeling. *ACM Trans. Graph.*, 2008, 27(5): 108:1–108:7.
- [11] Li C, Deussen O, Song Y Z, et al. Modeling and generating moving trees from video. *ACM Trans. Graph.*, 2011, 30(6): 127:1–127:12.
- [12] Verroust A, Rocquencourt I, Lazarus F. Extracting Skeletal Curves from 3D Scattered Data. *The Visual Computer*, 1999, 16: 15–25.
- [13] Bucksch A. SKELETONIZATION AND SEGMENTATION OF POINT CLOUDS USING OCTREES AND GRAPH THEORY, 2006.
- [14] Xu H, Gossett N, Chen B. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. Graph.*, 2007, 26(4).
- [15] Côté J F, Widlowski J L, Fournier R A, et al. The structural and radiative consistency of three-dimensional tree reconstructions from terrestrial lidar. *Remote Sensing of Environment*, 2009, 113(5): 1067 – 1081.
- [16] Horn B K P, Schunck B G. Determining Optical Flow. *ARTIFICIAL INTELLIGENCE*, 1981, 17: 185–203.
- [17] Lucas B D, Kanade T. An Iterative Image Registration Technique with an Application to Stereo Vision. 1981. 674–679.
- [18] Bouguet J. Pyramidal implementation of the Lucas Kanade feature tracker. Intel Corporation, Microprocessor Research Labs, 2000..
- [19] Liu J, Zhang X, Li H, et al. Creation of tree models from freehand sketches by building 3D skeleton point cloud. *Proceedings of the Entertainment for education, and 5th international conference on E-learning and games*, Berlin, Heidelberg: Springer-Verlag, 2010. 621–632.