



同濟大學  
TONGJI UNIVERSITY

## 工学硕士学位论文

# 你的标题

(国家自然科学基金 (No.XXXXXXXX) 支持)

姓 名：你的姓名

学 号：10XXXXXXXXXX

所在院系：电子与信息工程学院

学科门类：计算机科学与技术

学科专业：计算机应用技术

指导教师：你的教授

二〇一三年五月





同濟大學  
TONGJI UNIVERSITY

A dissertation submitted to  
Tongji University in conformity with the requirements for  
the degree of Master of Science

## Your title

(Supported by the Natural Science Foundation of China for  
Grant No.XXXXXXXX)

Candidate: Tongji Ren  
Student Number: 10XXXXXXXXX  
School/Department: School of Electrical and Informational Engineering  
Discipline: Computer Science and Technology  
Major: Computer Application Technology  
Supervisor: Prof. XXXXXX

May, 2013



# 学位论文版权使用授权书

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以盈利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：

年   月   日



# 同济大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年      月      日



## 摘 要

论文的摘要是对论文研究内容和成果的高度概括。摘要应对论文所研究的问题及其研究目的进行描述，对研究方法和过程进行简单介绍，对研究成果和所得结论进行概括。摘要应具有独立性和自明性，其内容应包含与论文全文同等量的主要信息。使读者即使不阅读全文，通过摘要就能了解论文的总体内容和主要成果。

论文摘要的书写应力求精确、简明。切忌写成对论文书写内容进行提要的形式，尤其要避免“第1章……；第2章……；……”这种或类似的陈述方式。

本文介绍同济大学论文模板 TONGJITHESIS 的使用方法。本模板符合学校的硕士、博士论文格式要求。

本文的创新点主要有：

- 用例子来解释模板的使用方法；
- 用废话来填充无关紧要的部分；
- 一边学习摸索一边编写新代码。

关键词是为了文献标引工作、用以表示全文主要内容信息的单词或术语。关键词不超过5个，每个关键词中间用分号分隔。（模板作者注：关键词分隔符不用考虑，模板会自动处理。英文关键词同理。）

**关键词：**T<sub>E</sub>X, L<sub>A</sub>T<sub>E</sub>X, CJK, 模板, 论文



## Abstract

An abstract of a dissertation is a summary and extraction of research work and contributions. Included in an abstract should be description of research topic and research objective, brief introduction to methodology and research process, and summarization of conclusion and contributions of the research. An abstract should be characterized by independence and clarity and carry identical information with the dissertation. It should be such that the general idea and major contributions of the dissertation are conveyed without reading the dissertation.

An abstract should be concise and to the point. It is a misunderstanding to make an abstract an outline of the dissertation and words “the first chapter”, “the second chapter” and the like should be avoided in the abstract.

Key words are terms used in a dissertation for indexing, reflecting core information of the dissertation. An abstract may contain a maximum of 5 key words, with semicolons used in between to separate one another.

**Key words:**  $\text{\TeX}$ ,  $\text{\LaTeX}$ , CJK, template, thesis

# 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 方法综述与技术简介 .....	3
2.1 树木建模方法综述 .....	3
2.2 基于多张图像的三维重建方法简介 .....	3
2.3 基于点云的骨架抽取方法综述 .....	3
第 3 章 基于图像的树木轻量化3D建模方法 .....	4
3.1 技术路线 .....	4
3.1.1 基于改进的PyrLK光流法的图像特征匹配 .....	4
3.1.2 三维重建 .....	4
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 .....	5
3.1.4 基于用户交互的模型改善与轻量化 .....	5
3.1.5 建模质量评估 .....	5
3.2 技术路线图 .....	5
第 4 章 基于图像树木轻量化建模的若干算法 .....	6
4.1 基于改进的PyrLK光流法的特征点匹配方法 .....	6
4.1.1 对于SIFT特征点匹配的尝试 .....	6
4.1.2 基于PyrLK光流法的特征点匹配 .....	6
4.1.3 改进的PyrLK光流法 .....	6
4.1.4 基于改进PyrLK光流法的三维重建 .....	6
4.2 基于三维体素泛洪与线性拟合的三维树木骨架抽取 .....	6
4.2.1 三维体素模型 .....	7
4.2.2 三维体素泛洪确定邻域范围 .....	7
4.2.3 通过最小二乘法线性拟合确定分支 .....	10
4.2.4 半径怎么探索？？ .....	10
4.3 基于枝干合并的轻量化处理 .....	10
4.3.1 L-System的尝试 .....	10
4.3.1.1 L-System简介 .....	10

4.3.1.2 树木模型的参数化L-System规则抽取 .....	11
4.3.1.3 使用L-System进行树木轻量化建模遇到的问题 .....	11
4.3.2 树木轻量化？枝干合并！ .....	12
4.4 建模质量评估算法 .....	12
4.5 本章小节 .....	12
第 5 章 实验过程与分析 .....	15
5.1 实验环境 .....	15
5.2 实验结果与分析 .....	15
第 6 章 总结与展望 .....	16
6.1 总结 .....	16
6.2 未来的工作 .....	16
参考文献 .....	17

# 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，并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出树木的骨架。
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，对于建模质量和轻量化过程中真实感的下降程度给出了函数化和量化的评价依据。

## 第 2 章 方法综述与技术简介

- 2.1 树木建模方法综述
- 2.2 基于多张图像的三维重建方法简介
- 2.3 基于点云的骨架抽取方法综述

## 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

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。然后再结合改进的PyrLK光流法，对找到的特征点在一定的容错区间进行3D匹配，最终将匹配结果存储到匹配文件以供后续使用。

#### 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。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

## 3.2 技术路线图

## 第4章 基于图像树木轻量化建模的若干算法

### 4.1 基于改进的PyrLK光流法的特征点匹配方法

#### 4.1.1 对于SIFT特征点匹配的尝试

对于特征点的匹配，本文首先尝试的是利用VisualSFM工具自带的SIFT特征点匹配。但是经过大量实验发现，该工具自带的SIFT特征点匹配并不能很完整地找到匹配的点对，从而导致了三维重建所依赖的数据不足、不精确，最后重建出来一个缺失度很大的模型，这样的模型显然不能成功的进行骨架抽取。

根据实验结果分析，除开

#### 4.1.2 基于PyrLK光流法的特征点匹配

#### 4.1.3 改进的PyrLK光流法

#### 4.1.4 基于改进PyrLK光流法的三维重建

### 4.2 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。

### 4.2.1 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。

图片。。。

伪代码。。。

### 4.2.2 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27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27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。

图片。。。

伪代码。。。

在进行三维泛洪的时候，可以编程实现27个方向迭代过程的并行化，以提高算法的效率。

---

**Algorithm 1** 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:   if  $P_x > X_{max}$  then
4:      $X_{max} = P_x$ 
5:   end if
6:   if  $P_y > Y_{max}$  then
7:      $Y_{max} = P_y$ 
8:   end if
9:   if  $P_z > Z_{max}$  then
10:     $Z_{max} = P_z$ 
11:   end if
12:   if  $P_x < X_{min}$  then
13:      $X_{min} = P_x$ 
14:   end if
15:   if  $P_y < Y_{min}$  then
16:      $Y_{min} = P_y$ 
17:   end if
18:   if  $P_z < Z_{min}$  then
19:      $Z_{min} = P_z$ 
20:   end if
21: end for
22: for all 空间点  $P(P_x, P_y, P_z) \in M$  do
23:    $V_x = \frac{P_x - X_{min}}{X_{max} - X_{min}} \cdot d$ 
24:    $V_y = \frac{P_y - Y_{min}}{Y_{max} - Y_{min}} \cdot d$ 
25:    $V_z = \frac{P_z - Z_{min}}{Z_{max} - Z_{min}} \cdot d$ 
26:    $\mathbb{V}[V_x, V_y, V_z] = \mathbb{V}[V_x, V_y, V_z] \cup \{P\}$ 
27: end for
```

---

---

**Algorithm 2** 三维体素泛洪确定邻域范围

---

**Input:** 当前体素 $C$ **Input:** 三维体素数组 $\mathbb{V}[1..d, 1..d, 1..d]$ **Input:** 泛洪方向数组 $\mathbb{D}[1..27]$ **Input:** 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  $\mathbb{S}$ 

```

---

### 4.2.3 通过最小二乘法线性拟合确定分支

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27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线段，作为该部分的骨架。

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 $d_1, d_2 \dots d_n$ ，分别对它们进行基于最小二乘法的线性拟合，这个步骤可以实现为并行以提升计算的速度。具体方法是：对某个方向 $d_i$ ，将其包含的体素中的点全部抽取出来，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：

### 4.2.4 半径怎么探索？？

## 4.3 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。

### 4.3.1 L-System的尝试

#### 4.3.1.1 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。

#### 4.3.1.2 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：

$$N(l, r) \rightarrow \&(\theta_0)\wedge(\phi_0)!r+(l)S_0(l*a_0, r*b_0)\dots\&(\theta_n)\wedge(\phi_n)!r+(l)S_n(l*a_n, r*b_n) \quad (4.1)$$

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。

#### 4.3.1.3 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4.1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。

#### 4.3.2 树木轻量化？枝干合并！

用L-System的方法抽取规则所产生的问题，从本质上看，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。

如图所示。。。进行图片阐述。。。

伪代码。。。

功能：根据纵向合并角度参数，以当前节点为发起点递归式地纵向合并枝干

功能：根据横向合并角度参数，横向合并夹角小于角度阈值的末端叶子枝干

### 4.4 建模质量评估算法

### 4.5 本章小节

---

**Algorithm 3** 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

```

---

---

**Algorithm 4** 横向合并枝干

---

**Input:** 初始化横向合并角度 $\beta$

**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:      $\vec{u} \leftarrow N_1.Position - N.Position$ 
6:      $\vec{v} \leftarrow N_2.Position - N.Position$ 
7:      $\gamma \leftarrow \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|}\right)$ 
8:     if  $\gamma < \beta$  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
```

---

## 第 5 章 实验过程与分析

5.1 实验环境

5.2 实验结果与分析

## 第 6 章 总结与展望

6.1 总结

6.2 未来的工作

## 参考文献