OOP Team Project 设计文档

周先达, 孟垂正

2016 年 6 月 26 日

1 问题简介

本问题为 Team Project 中的 Problem3: 3D Placement Using the T-tree Formulation, 即根据参考文献 [1] 中提出的 T-tree 结构和模拟退火算法解决以下问题: 给定一定数量的三维尺寸已知的长方体, 求出一种使得其包围盒体积最小的布置方案. 这一问题的求解方法可用于解决 FPGA 上的集成电路布线问题.

2 系统总览

本程序可以从文件读入有限数量的长方体的三维尺寸, 通过 T-tree 结构和模拟退火算法计算出包围盒体积较小, 空间利用率较高的布置方案并存储到文件, 布置方案可以通过 OpenGL 进行展示.

3 系统结构

3.1 模块划分

系统可划分为以下模块:

- T-tree 结构的实现
- 模拟退火算法
- 结果显示

其中 T-tree 结构实现部分由孟垂正完成, 模拟退火算法和结果显示部分由周先达完成.

3.2 文件结构

- Ttree.h: 声明了程序中所需数据结构
- Ttree.cpp: 实现了各数据结构的操作和模拟退火算法
- Display.h: 实现了 OpenGL 显示功能
- Ttreetest.cpp: 主函数

4 数据设计

4.1 Point3d

3D 点类, 包括该点的 x, y, t 坐标.

4.2 Task

存储代表各个 Task 的长方体数据, 包括长方体 x, y, t 均最小的一点的坐标 (题目要求长方体平行坐标轴放置) 和长方体的长宽高.

4.3 TtreeNode

用于 Ttree 的节点类, 存储了某个 Task 的指针及该节点的父亲与 3 个孩子.

4.4 Ttree

根据参考文献实现的 Ttree 类, 用于表示所有长方体的位置关系. 实现了从 Ttree 到各长方体精确坐标的转换.

4.5 Contour

根据参考文献实现的 Contour 类, 在 Ttree 转换为长方体坐标时用于确定 y 方向坐标, 与枚举方法相比 Contour 可做到 O(1) 的时间复杂度 [2].

5 模块设计

5.1 T-tree 结构

5.1.1 T-tree 的构建

T-tree 结构用于表示长方体之间的相互位置关系. 对于任意 2 个长方体 A 和 B, 其间的位置关系可以分为 3 类 (用 Ao 和 Bo 表示二者 x, y, t 均最小一点的坐标, X, Y, T 分别表示长方体的三维尺寸):

1. Ao 位于 Bo+T 的位置. 此时 A 为 B 的左孩子.
2. Ao 与 Bo 的 T 位置相同, Ao 的 Y 坐标大于 Bo. 此时 A 为 B 的中孩子
3. Ao 与 Bo 的 T, Y 位置相同, Ao 的 X 坐标大于 Bo. 此时 A 为 B 的右孩子.

根据以上方法可以构建一棵 T-tree.

5.1.2 T-tree 转换为长方体坐标

采用拆分的方法可以将 T-tree 转换为多个二叉树, 每个二叉树在确定其根节点坐标后均可确定其所有孩子的坐标. 本部分功能由`double Ttree::pack()`实现, 该函数在确定所有长方体坐标之后返回包含所有长方体的包围盒体积.

已知父节点坐标时子节点坐标的确定

1. T 坐标. 根据定义, T 坐标可直接进行确定: 若为左孩子则取父节点的 T 坐标 + 父节点 T 尺寸, 否则与父节点 T 坐标相同. 在算法中, T 坐标可以最先确定.

2. Y 坐标. 已知 T 坐标时, 可在 Contour 结构 (见下文) 中以 O(1) 时间确定 Y 坐标.

3. X 坐标. 已知 T, Y 坐标时, 遍历已被放置的所有长方体, 计算与待放置长方体相冲突的所有长方体的最大 X 坐标, 待放置长方体在 X 方向上不能与它们冲突.

T-tree 拆分为二叉树 使用一个 stack 存储拆分得到的二叉树根节点. 从 T-tree 的根节点出发, 若当前节点有右孩子, 则将其放入 stack, 然后将当前节点视为二叉树根节点执行另一子程序.

二叉树中节点位置的确定 依照 DFS 顺序遍历二叉树, 根据上文提及的方法可以确定所有节点的位置. 注意遍历过程中应当将节点的右孩子 (如果存在) 加入上文的 stack 中. 遍历过程中, T-tree 的左孩子为二叉树的左孩子, T-tree 的中孩子为二叉树的右孩子.

5.1.3 用于快速确定 Y 方向坐标的 Contour 结构

Contour 结构实际上是一个存储了当前位于边界的边的链表, 如图所示:

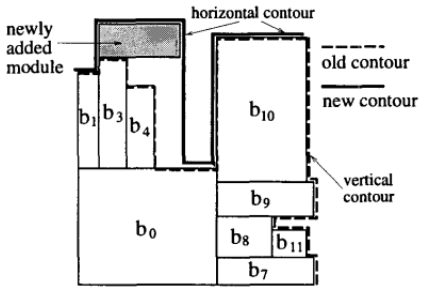

Figure 1: Contour 结构示意图

需要确定新节点的 Y 坐标时, 只需在 Contour 中查找与之可能冲突的边并找到其最大 Y 坐标即可, 无需对所有块进行遍历. 插入后, 将被覆盖的边替换为新节点对应的边即可.

5.1.4 T-tree 的变换

模拟退火算法要求 T-tree 能够通过一些变换转换为另一个 T-tree, 这里实现了论文中提到的三种变换:

1. 节点移动. 随机选择一个节点并将其从树中摘除, 然后在树中随机选择一个位置将其插入.

2. 节点交换. 随机选择 2 个节点进行交换.

3. 节点旋转. 随机选择一个节点, 将其代表的长方体的 X 尺寸和 Y 尺寸交换, 相当于在 X-Y 平面上进行旋转.

5.2 模拟退火算法

试验时，起始温度为 1，终止温度 10^{-8}。一个测例 100 个随机生成的长宽高在 [1, 10] 内的立方块，大约在 5-6 分钟跑完，能达到 80-85% 的空间利用率

5.2.1 温度下降方法/算法终止准则

采用等比例下降的方法，有一个优化是当前 100 次降温得到相同的空间利用率（即 100 次降温没有接受一个新的 T 树）时，对算法进行升温，升温的幅度相当于 50 次降温的幅度。这种做法更多可以看作是延缓降温，期待 T 树能有所变化。最后当 1000 次降温都没有接受新树时，说明已经达到了一个难以跳出的局部最优，算法直接终止，不再等待算法终止。

5.2.2 每一温度下停止准则

每个温度下，T 树变化的次数与 T 树结点个数和温度相关。
低温时，空间利用率较高而状态接受率较低，为鼓励在低温时更多的接受，我们让每个温度下的变化次数随温度改变，温度越高变化次数越多。
具体变化由函数 `gettime()` 决定，`gettime()` 返回一个跟当前温度二次相关的系数，这个系数乘以 T 树的结点个数就是该温度下该 T 树的变化次数。

5.3 结果显示

5.3.1 交互方法与显示

按鼠标左键拖动缩放
按鼠标右键拖动旋转
按 l 键切换显示模式
按 esc 键退出

在线框显示模式下，用白线画出所有立方体的所有边，粉线画出整个 Packing 的边框。在面显示模式下，用灰线画出所有立方体的 Packing 的边框，立方体面是随光源变换的颜色的半透明面，另外，在当前视角下重叠越多的地方越亮。

5.3.2 基本的绘制

光源和材质的设置

```
glMaterialfv(GL_FRONT,GL_AMBIENT,matAmbient);
glMaterialfv(GL_FRONT,GL_DIFFUSE,matDiff);
glLightfv(GL_LIGHT0,GL_AMBIENT,ambientLight);
glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuseLight);
glLightfv(GL_LIGHT0,GL_POSITION,lightPosition);
```

防锯齿

```
glEnable(GL_LINE_SMOOTH);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glHint(GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
```

逐个读取 output 文件中的 Task 位置和尺寸。在绘制回调函数中逐个绘制每个立方体的边和面。

```
    glEnable(GL_LIGHT0);
    for (int i = 0; i < tasknum; ++i) {
        drawFaceBlock(taskvec[i]);
    }
    glDisable(GL_LIGHT0);
    for (int i = 0; i < tasknum; ++i) {
        drawWireBlock(taskvec[i]);
    }
    drawContainer();
```

5.3.3 交互

窗口自适应

鼠标控制用 gluLookAt(eye, center, up) 确定视角，需要确定三个点，eye，center，up。此处给定 up 值恒定（0，1，0）。需要两个位置 eye，center 的坐标。而前后移动是移动 center。然后利用球坐标系，通过 center 来求得 eye，最后 gluLookAt 就行了，旋转就是在增大或减小球坐标系中的两个角度值。

```
void CalEyePosition() {
    if(yrotate > PI) yrotate = PI; // 限制旋转方向
    if(yrotate < 0.01)  yrotate = 0.01;
    if(xrotate > 2*PI)   xrotate = 0.01;
    if(xrotate < 0)   xrotate = 2 * PI;
    if(celength > 50)  celength = 50; // 限制缩放大小
    if(celength < 5)   celength = 5;
    eye[0] = center[0] + celength * sin(yrotate) * cos(xrotate);
    eye[2] = center[2] + celength * sin(yrotate) * sin(xrotate);
    eye[1] = center[1] + celength * cos(yrotate);
}
```

鼠标控制函数，其中的常数控制缩放/旋转的速率。

```
void MouseMotion(int x, int y) {
    if(mouserdown == GL_TRUE) {
        xrotate += (x - mousex) / 80.0f;
        yrotate -= (y - mousey) / 120.0f;
    }

    if(mouseldown == GL_TRUE) {
        celength += (y - mousey) / 25.0f;
    }
    mousex = x, mousey = y;
    CalEyePosition();
    glutPostRedisplay();
}
```

键盘控制

```
void processNormalKeys(unsigned char key, int x, int y) {
    if (key == 27)
        exit(0);
    if (key == 'l') {
        GLint params[] = {GL_POLYGON_MODE, GL_FILL};
        glGetIntegerv(GL_POLYGON_MODE, params);
        if (*(params) == GL_FILL) {
            glPolygonMode(GL_FRONT_AND_BACK ,GL_POINT);
            glDisable(GL_LIGHTING);
```

```
        }
        else {
            glPolygonMode(GL_FRONT_AND_BACK ,GL_FILL);
            glEnable(GL_LIGHTING);
        }
    }
}
```

因为 glut 的主显示循环不能 return，需要有含方法令其推出，key == 27 表示 esc 键。点按 l 切换多边形的显示方式并开关光效，从而在两种显示模式之间切换。

6 使用方法

有两种运行模式，编译后运行`./main.out <起始温度> <终止温度> <降温比例>`，运行事先准备好的 10 个测例，将数据输出到 output 文件夹中。
运行`./main.out <测例文件名> <输出文件名> <起始温度> <终止温度> <降温比例>`，运行某个指定的测例，最后会绘制结果。

7 运行结果

实验次数	立方体个数	立方体总体积	边界盒体积	体积利用率 (%)	运行时间 (s)
1	84	12293	15000	81.95	335.16
2	88	16127	19400	83.13	374.61
3	99	12525	15100	82.95	427.79
4	81	15311	19000	80.58	297.94
5	98	15186	17900	84.84	470.01
6	91	14297	17200	83.12	363.12
7	84	12293	14900	82.50	329.51
8	97	17685	21600	81.88	427.87
9	82	14744	17600	83.77	338.38
10	88	15307	18700	81.86	373.09

参考文献

[1] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang, "Temporal floorplanning using the t-tree formulation," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pp. 300–305, IEEE Computer Society, 2004.

[2] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-trees: a new representation for non-slicing floorplans," in *Proceedings of the 37th Annual Design Automation Conference*, pp. 458–463, ACM, 2000.