|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **研发生产中心** | **文档编号** |  | **版本** | A1 | **密级** | 商密A |
| **项目名称** | **金属三维打印数据处理软件系统研究与开发** | | | | |
| **项目来源** |  | | | | |

QR-RD-022(Ver1.2)

**Mesh print系统**

**详细设计说明书**

(内部资料 请勿外传)

|  |  |  |  |
| --- | --- | --- | --- |
| **编 写：** |  | **日 期：** |  |
| **检 查：** |  | **日 期：** |  |
| **审 核：** |  | **日 期：** |  |
| **批 准：** |  | **日 期：** |  |

**XX公司**

**版权所有 不得复制**

**文档变更记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **变更（+/-）说明** | **作者** | **版本号** | **日期** | **批准** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目 录

[1. 引言 4](file:///D:\模板\详细设计.doc#_Toc266115737#_Toc266115737)

[1.1 编写目的和范围 4](file:///D:\模板\详细设计.doc#_Toc266115738#_Toc266115738)

[1.2 术语表 4](file:///D:\模板\详细设计.doc#_Toc266115739#_Toc266115739)

[1.3 参考资料 4](file:///D:\模板\详细设计.doc#_Toc266115740#_Toc266115740)

[1.4 使用的文字处理和绘图工具 4](file:///D:\模板\详细设计.doc#_Toc266115741#_Toc266115741)

[2. 全局数据结构说明 5](file:///D:\模板\详细设计.doc#_Toc266115742#_Toc266115742)

[2.1 常量 5](file:///D:\模板\详细设计.doc#_Toc266115743#_Toc266115743)

[2.2 变量 5](file:///D:\模板\详细设计.doc#_Toc266115744#_Toc266115744)

[2.3 数据结构 5](file:///D:\模板\详细设计.doc#_Toc266115745#_Toc266115745)

[3. 模块设计 5](file:///D:\模板\详细设计.doc#_Toc266115746#_Toc266115746)

[3.1 用例图 5](file:///D:\模板\详细设计.doc#_Toc266115747#_Toc266115747)

[3.2 功能设计说明 5](file:///D:\模板\详细设计.doc#_Toc266115748#_Toc266115748)

[3.2.1 模块1 5](file:///D:\模板\详细设计.doc#_Toc266115749#_Toc266115749)

[3.2.2 模块2 6](file:///D:\模板\详细设计.doc#_Toc266115750#_Toc266115750)

[4. 接口设计 7](file:///D:\模板\详细设计.doc#_Toc266115751#_Toc266115751)

[4.1 内部接口 7](file:///D:\模板\详细设计.doc#_Toc266115752#_Toc266115752)

[4.2 外部接口 7](file:///D:\模板\详细设计.doc#_Toc266115753#_Toc266115753)

[4.2.1 接口说明 7](file:///D:\模板\详细设计.doc#_Toc266115754#_Toc266115754)

[4.2.2 调用方式 8](file:///D:\模板\详细设计.doc#_Toc266115755#_Toc266115755)

[5. 数据库设计 8](file:///D:\模板\详细设计.doc#_Toc266115756#_Toc266115756)

[6. 系统安全保密设计 8](file:///D:\模板\详细设计.doc#_Toc266115757#_Toc266115757)

[6.1 说明 8](file:///D:\模板\详细设计.doc#_Toc266115758#_Toc266115758)

[6.2 设计 8](file:///D:\模板\详细设计.doc#_Toc266115759#_Toc266115759)

[6.2.1 数据传输部分 8](file:///D:\模板\详细设计.doc#_Toc266115760#_Toc266115760)

[6.2.2 IP过滤分部 8](file:///D:\模板\详细设计.doc#_Toc266115761#_Toc266115761)

[6.2.3 身份验证部分 8](file:///D:\模板\详细设计.doc#_Toc266115762#_Toc266115762)

[7. 系统性能设计 9](file:///D:\模板\详细设计.doc#_Toc266115763#_Toc266115763)

[8. 系统出错处理 9](file:///D:\模板\详细设计.doc#_Toc266115764#_Toc266115764)

# 引言

## 编写目的和范围

本详细设计说明书编写的目的是说明程序模块的设计考虑，包括程序描述、输入/输出、算法和流程逻辑等，为软件编程和系统维护提供基础。本说明书的预期读者为系统设计人员、软件开发人员、软件测试人员和项目评审人员。

## 术语表

|  |  |  |
| --- | --- | --- |
| **序号** | **术语或缩略语** | **说明性定义** |
| 1 | PM | Project Manager,项目经理 |
| 2 |  |  |
|  |  |  |

## 参考资料

|  |  |  |  |
| --- | --- | --- | --- |
| 资料名称 | 作者 | 文件编号、版本 | 资料存放地点 |
| 一种表示流形形体和非流形形体的统一数据结构 | 张成林，文姗姗，朱琳 |  |  |
| 面向金属三维打印工艺的模型内部支撑结构功能软件 |  | 2017SR023381 |  |
| 面向金属三维打印工艺的数据处理功能软件 |  | 2017SR066059 |  |

## 使用的文字处理和绘图工具

文字处理软件：编写设计文档使用的文字处理软件：Microsoft Office 2016。

绘图工具：使用的UML工具：Visio

# 全局数据结构说明

## 常量

## 变量

程序的所有全局变量声明在globalFunction.h中，。

1. **extern** **float** field\_width\_;
2. **extern** **float** field\_height\_ ;
3. **extern** **float** line\_width\_ ;
4. **extern** **float** field\_overlap\_ ;
5. **extern** **float** unit ;
6. **extern** **int** units\_y\_;
7. **extern** **int** units\_x\_;
8. **extern** **float** thickness\_;
9. **extern** **float** offset\_dis\_;
10. **extern** **int** \* num\_hatch;
11. **extern** **float** DEFAULT\_L ;
12. **extern** **float** THRESHOLD;
13. **extern** **float** THRESHOLD1;
14. **extern** **float** GAP;
15. **extern** **float** SEGLENGTH ;
16. **extern** **float** RESO ;
17. **extern** **float** VERTICALGAP;
18. **extern** **float** laser\_power\_hatch\_ ;
19. **extern** **float** laser\_speed\_hatch\_ ;
20. **extern** **float** laser\_power\_polygon\_ ;
21. **extern** **float** laser\_speed\_polygon\_ ;
22. **extern** **int** increment\_angle\_;
23. **extern** **float** scaleV;
24. **extern** **float** scaleT;
25. **extern** **int** sss;
26. **extern** **int** fildID;

## 数据结构

Meshprint系统采用半边数据结构来存储模型的拓扑信息。半边数据结构不会存储网格的边的信息，取而代之的是半边(half-edges)。半边是有方向的，并且一条边的一对半边有相反的方向。下图展示了三角形网格的一小部分的半边描述。蓝色是顶点，橙色是半边，其中的箭头表示指针。

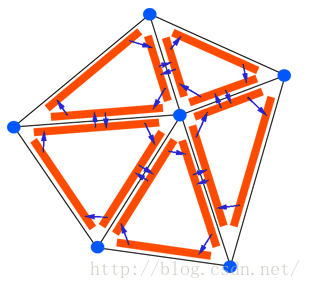


图 1 半边数据结构

围绕一个面的三条边形成了一个环(circular linked list)。这个链可以顺时针也可以逆时针，只要在使用中保持一致就好。环中的每半边存储一个指向以其为边的面的指针、指向半边终点的指针和指向它的另一条半边的指针。

# 模块设计

## 用例图

## 功能设计说明

本项目采用面向对象的编程技术，对快速原型数据预处理软件进行总体设计，系统由开发平台及应用程序两部分组成， 在WindowsXP操作系统下，基于OpenGL图形库，利用Microsoft Visual C++集成开发环境进行开发。 快速原型制造数据预处理软件模块结构图如下。

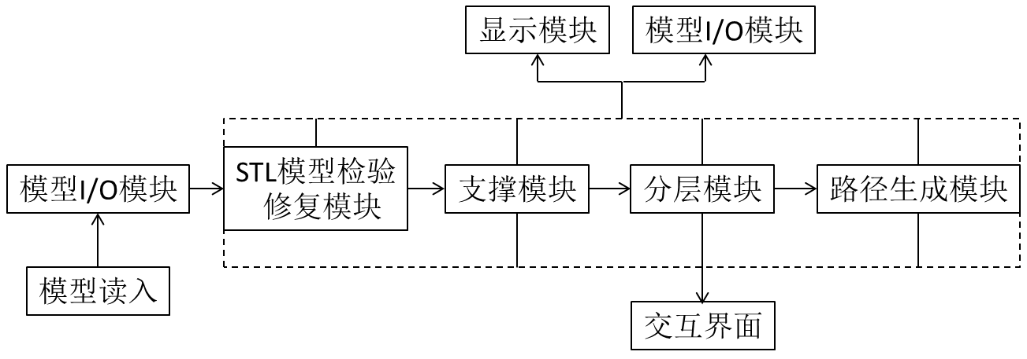
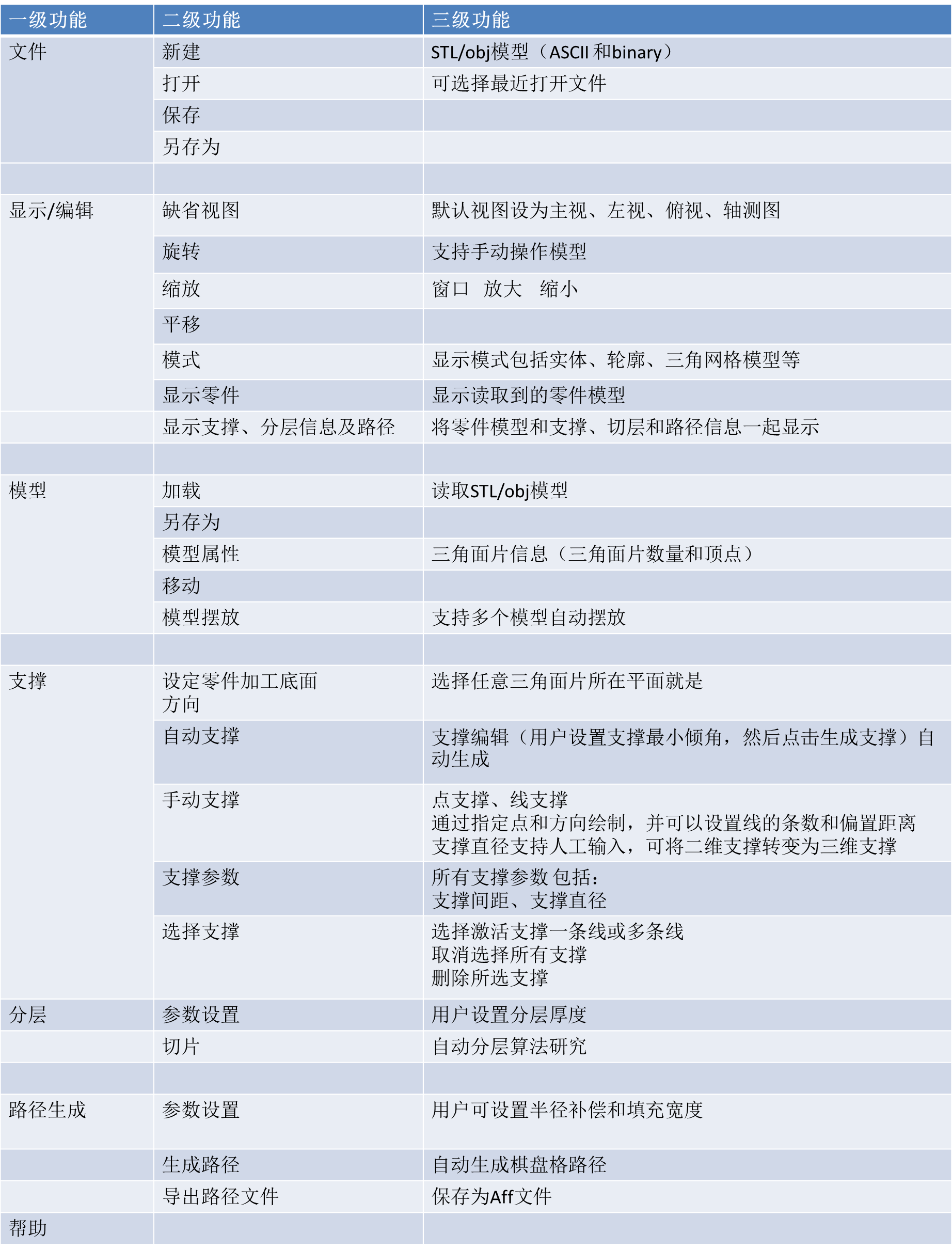


图 2软件模块结构图

具体功能定义如图3



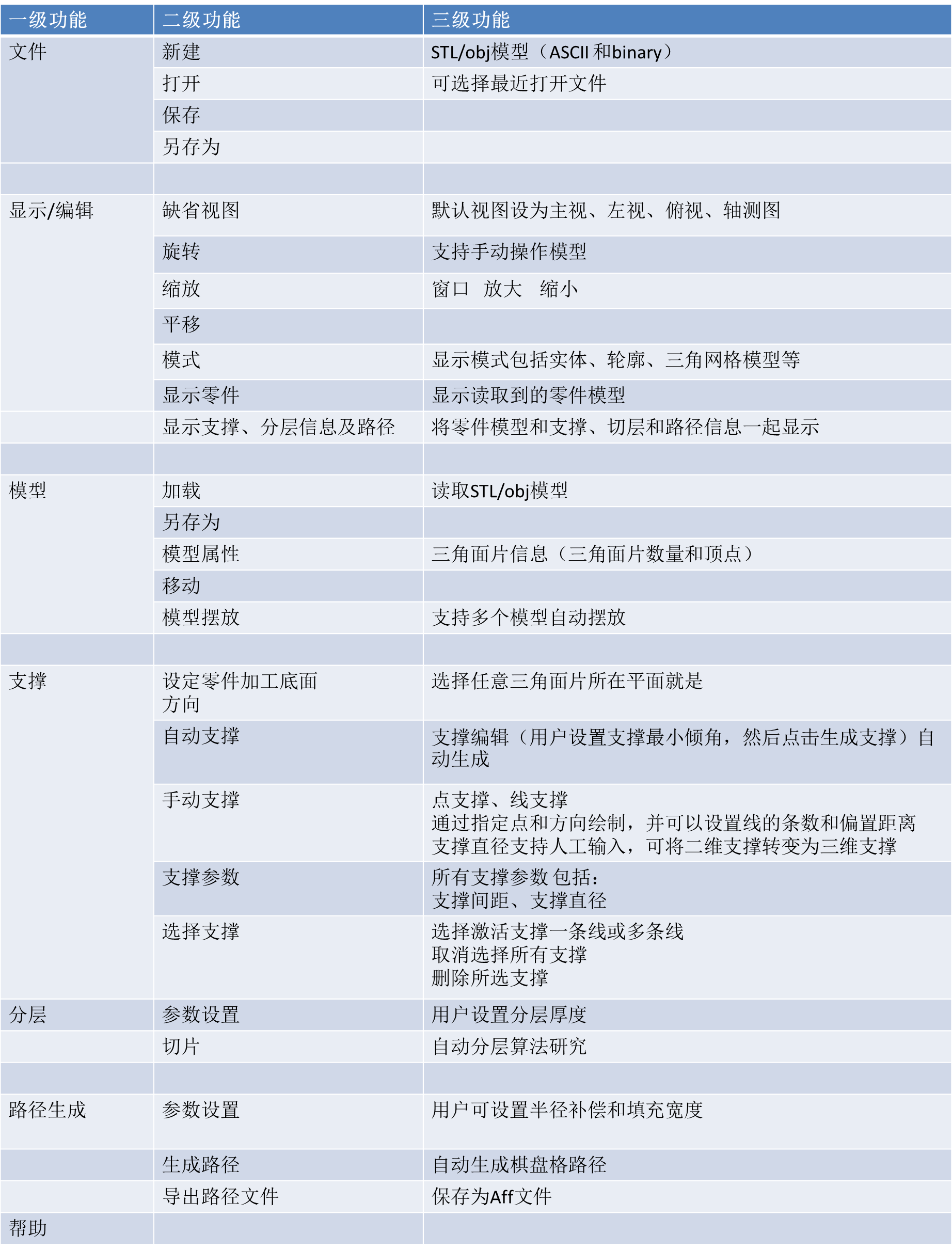


图 3 软件功能定义

### 模型导入模块

模型输入读取模块主要包含OBJ文件读取子模块、STL文件读取子模块两部分。

#### 模型读取子模块

##### 功能描述

该模块将OBJ、STL文件中的点、面、位置坐标以及法向信息读入内存，建立相邻信息点的拓扑结构。

##### 输入数据

输入的数据为obj格式和STL格式的文件。OBJ文件是Alias|Wavefront公司为它的一套基于工作站的3D建模和动画软件"Advanced Visualizer"开发的一种标准3D模型文件格式，很适合用于3D软件模型之间的互导。

（1）OBJ文件不需要任何种文件头(File Header)，尽管经常使用几行文件信息的注释作为文件的开头。OBJ文件由一行行文本组成，注释行以符号“#”为开头，空格和空行可以随意加到文件中以增加文件的可读性。有字的行都由一两个标记字母也就是关键字(Keyword)开头，关键字可以说明这一行是什么样的数据。多行可以逻辑地连接在一起表示一行，方法是在每一行最后添加一个连接符(\)。注意连接符(\)后面不能出现空格或Tab格，否则将导致文件出错。

下列关键字可以在OBJ文件使用。在这个列表中, 关键字根据数据类型排列，每个关键字有一段简短描述。

1. 顶点数据(Vertex data)：
2. v 几何体顶点(Geometric vertices)
3. vt 贴图坐标点(Texture vertices)
4. vn 顶点法线(Vertex normals)
5. vp 参数空格顶点 (Parameter space vertices)
6. 自由形态曲线(Free-form curve)/表面属性(surface attributes):
7. deg 度(Degree)
8. bmat 基础矩阵(Basis matrix)
9. step 步尺寸(Step size)
10. cstype 曲线或表面类型 (Curve or surface type)
11. 元素(Elements):
12. p 点(Point)
13. l 线(Line)
14. f 面(Face)
15. curv 曲线(Curve)
16. curv2 2D曲线(2D curve)
17. surf 表面(Surface)
18. 自由形态曲线(Free-form curve)/表面主体陈述(surface body statements):
19. parm 参数值(Parameter values )
20. trim 外部修剪循环(Outer trimming loop)
21. hole 内部整修循环(Inner trimming loop)
22. scrv 特殊曲线(Special curve)
23. sp 特殊的点(Special point)
24. end 结束陈述(End statement)
25. 自由形态表面之间的连接(Connectivity between free-form surfaces):
26. con 连接 (Connect)
27. 成组(Grouping):
28. g 组名称(Group name)
29. s 光滑组(Smoothing group)
30. mg 合并组(Merging group)
31. o 对象名称(Object name)
32. 显示(Display)/渲染属性(render attributes):
33. bevel 导角插值(Bevel interpolation)
34. c\_interp 颜色插值(Color interpolation)
35. d\_interp 溶解插值(Dissolve interpolation)
36. lod 细节层次(Level of detail)
37. usemtl 材质名称(Material name)
38. mtllib 材质库(Material library)
39. shadow\_obj 投射阴影(Shadow casting)
40. trace\_obj 光线跟踪(Ray tracing)
41. ctech 曲线近似技术(Curve approximation technique)
42. stech 表面近似技术 (Surface approximation technique)

（2）STL是最多[快速原型](http://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E5%8E%9F%E5%9E%8B)系统所应用的标准文件[类型](http://baike.baidu.com/item/%E7%B1%BB%E5%9E%8B/33084)。STL是用三角网格来表现3D CAD模型。

在STL文件中的三角面片的信息单元 facet 是一个带矢量方向的三角面片，STL三维模型就是由一系列这样的三角面片构成。

整个STL文件的首行给出了文件路径及文件名。

在一个 STL文件中，每一个facet由7 行数据组成，

facet normal 是三角面片指向实体外部的法矢量坐标，

outer loop 说明随后的3行数据分别是三角面片的3个顶点坐标，3顶点沿指向实体外部的法矢量方向逆时针排列。

ASCII码格式的STL文件逐行给出三角面片的几何信息，每一行以1个或2个关键字开头。ASCII格式的STL 文件结构如下：

1. 明码://字符段意义
2. solidfilenamestl//文件路径及文件名
3. facetnormalxyz//三角面片法向量的3个分量值
4. outerloop
5. vertexxyz//三角面片第一个顶点坐标
6. vertexxyz//三角面片第二个顶点坐标
7. vertexxyz//三角面片第三个顶点坐标
8. endloop
9. endfacet//完成一个三角面片定义
11. ......//其他facet
13. endsolidfilenamestl//整个STL文件定义结束

二进制STL文件用固定的字节数来给出三角面片的几何信息。文件起始的80个字节是文件头，用于存贮文件名；紧接着用 4 个字节的整数来描述模型的三角面片个数，后面逐个给出每个三角面片的几何信息。每个三角面片占用固定的50个字节，依次是:

3个4字节浮点数(角面片的法矢量)

3个4字节浮点数(1个顶点的坐标)

3个4字节浮点数(2个顶点的坐标)

3个4字节浮点数(3个顶点的坐标)个

三角面片的最后2个字节用来描述三角面片的属性信息。

一个完整二进制STL文件的大小为三角形面片数乘以 50再加上84个字节。

二进制格式的STL 文件结构如下:

1. UINT8//Header//文件头
2. **UINT32**//Numberoftriangles//三角面片数量
3. //foreachtriangle（每个三角面片中）
4. REAL32[3]//Normalvector//法线矢量
5. REAL32[3]//Vertex1//顶点1坐标
6. REAL32[3]//Vertex2//顶点2坐标
7. REAL32[3]//Vertex3//顶点3坐标
8. UINT16//Attributebytecountend//文件属性统计

##### 输出数据

详细描述子功能1所产生的数据以及这些数据的表现形式。

该子模块输出的数据正是后续模块操作的输入，即模型点线面的三维拓扑结构信息，是以半边数据结构形式存储在内存中的数据。

##### 算法和流程

从业务角度详细描述根据输入数据产生输出数据的业务算法和流程。

1. 利用C++的文件流类，读取文中所有点的坐标，针对OBJ文件中的每一行，构造一个HE\_Vert 对象，并将其地址存储到Mesh3D对象ptr\_mesh\_中的成员链表pvertices\_list\_中。
2. Obj文件的后一部分提供了三角面片的标号信息。根据标号信息找到pvertices\_list\_中对应的点，且每个三角面片由三个顶点组成，其中按逆时针方向，前后两点依次匹配，共组成三条有向线段。将三条有向线段构造三个HE\_Edge对象，按顺序存储到ptr\_mesh的pedge\_list\_中。同时完成对三条半边的指向点和指向面的明确。每个面片完成以上操作后，构造一个HE\_Face对象，存储到ptr\_mesh\_的pface\_list中。

##### 数据设计

给出本程序中的局部数据结构说明，包括数据结构名称，功能说明，具体数据结构说明（定义、注释设计、取值）等。相关数据库表，数据存储设计（具体说明需要以文件方式保存的数据文件名、数据存储格式、数据项及属性等。）

半边数据结构中需要对点，线，面做出定义，在程序中分别以HE\_Vert、HE\_Edge、HE\_Face作为存储形式。而Mesh3D是对模型位置及其拓扑信息完全表示的类，每个导入或者需要处理的模型，都是一个Mesh3D类的对象。下表清楚的列出了这四个类里面的成员变量，变量属性，成员函数以及成员函数的属性。半边数据结构的所有功能在Mesh3D.h和Mesh3D.cpp及vec.h三个文件中实现。

1. #pragma once
2. #includes
3. #include <vector>
4. #include <map>
5. #include "Vec.h"
6. #include <set>
7. Forward declarations
8. HE\_vert;
9. HE\_edge;
10. HE\_face;
11. **using** trimesh::point;
12. point;
13. Vec2f;
14. Vec3f;
15. Vec4f;
16. SelectTag
17. BoundaryTag
18. HE\_vert
19. **public**:
20. id\_;
21. id\_in\_STL;
22. position\_;
23. normal\_;
24. texCoord\_;
25. color\_;
26. pedge\_;
27. degree\_;
28. boundary\_flag\_;
29. selected\_;
30. adjHEdges;
31. neighborIdx;
32. pprev;
33. pnext;
34. helper;
35. **public**:
36. HE\_vert(**const** Vec3f& v) : id\_(-1), position\_(v), pedge\_(NULL), degree\_(0), boundary\_flag\_(INNER), selected\_(UNSELECTED) , color\_(255.f / 255.f, 215.f / 255.f, 0.f/ 255.f, 1.f),neighborIdx()
37. ~HE\_vert(**void**) { neighborIdx.clear();}
38. isOnBoundary(**void**) {**return** boundary\_flag\_==BOUNDARY;}
39. id(**void**) {**return** id\_;}
40. normal(**void**) {**return** normal\_;}
41. position(**void**) {**return** position\_;}
42. texCoordinate(**void**) {**return** texCoord\_;}
43. color(**void**) {**return** color\_;}
44. degree(**void**) {**return** degree\_;}
45. selected(**void**) {**return** selected\_;}
46. boundary\_flag(**void**) {**return** boundary\_flag\_;}
47. set\_normal(**const** Vec3f& n) {normal\_=n;}
48. set\_position(**const** Vec3f& p) {position\_=p;}
49. set\_id(**int** id) {id\_=id;}
50. set\_color(**const** Vec4f& c) {color\_=c;}
51. set\_seleted(**int** tag){selected\_=tag;}
52. set\_boundary\_flag(BoundaryTag bt) {boundary\_flag\_=bt;}
53. HE\_edge
54. **public**:
55. id\_;
56. pvert\_;
57. start\_;
58. ppair\_;
59. pface\_;
60. pnext\_;
61. pprev\_;
62. texCoord\_;
63. boundary\_flag\_;
64. is\_selected\_;
65. left\_point\_;
66. insert\_point\_;
67. helperL;
68. helperR;
69. **public**:
70. HE\_edge() : id\_(-1), pvert\_(NULL), ppair\_(NULL) , pface\_(NULL), pnext\_(NULL), pprev\_(NULL), boundary\_flag\_(INNER), is\_selected\_(**false**)
71. ~HE\_edge()
72. id(**void**) {**return** id\_;}
73. isBoundary(**void**) {**return** boundary\_flag\_==BOUNDARY;}
74. boundary\_flag(){**return** boundary\_flag\_;}
75. set\_boundary\_flag(BoundaryTag bt) {boundary\_flag\_ = bt;}
76. HE\_face
77. **public**:
78. id\_;
79. pedge\_;
80. normal\_;
81. valence\_;
82. selected\_;
83. color\_;
84. boundary\_flag\_;
85. com\_flag;
86. **public**:
87. HE\_face() : id\_(-1), pedge\_(NULL), valence\_(0), selected\_(UNSELECTED), boundary\_flag\_(INNER), normal\_(0,0,0),com\_flag(-1)
88. ~HE\_face()
89. id(**void**) {**return** id\_;}
90. valence(**void**) {**return** valence\_;}
91. normal(**void**) {**return** normal\_;}
92. selected(**void**) {**return** selected\_;}
93. color(**void**) {**return** color\_;}
94. boundary\_flag() {**return** boundary\_flag\_;}
95. set\_selected(**int** tag){selected\_=tag;}
96. set\_color(**const** Vec4f& c) {color\_=c;}
97. set\_boundary\_flag(BoundaryTag bt) {boundary\_flag\_=bt;}
98. face\_verts(std::vector<HE\_vert \*>& verts)
99. center()
100. **public**:
101. getBoundingBox(Vec3f& max\_out, Vec3f& min\_out)
102. comVertex
103. operator ()(HE\_vert\* a, HE\_vert\* b)**const**
104. Mesh3D
105. VERTEX\_ITER;
106. FACE\_ITER;
107. EDGE\_ITER;
108. VERTEX\_RITER;
109. FACE\_RITER;
110. EDGE\_RITER;
111. PAIR\_VERTEX;
112. **private**:
113. pvertices\_list\_;
114. pedges\_list\_;
115. pfaces\_list\_;
116. bheList;
117. iheList;
118. bLoop;
119. no\_loop;
120. input\_vertex\_list\_;
121. num\_components\_;
122. average\_edge\_length\_;
123. xmax\_, xmin\_, ymax\_, ymin\_, zmax\_, zmin\_;
124. wro\_Nor\_facets\_;
125. **public**:
126. edgemap\_;
127. Mesh3D(**void**);
128. ~Mesh3D(**void**);
129. exportNeighborId();
130. get\_vertex\_list(**void**) {**return** pvertices\_list\_;}
131. get\_edges\_list(**void**) {**return** pedges\_list\_;}
132. get\_bedges\_list(**void**) { **return** bheList; }
133. get\_faces\_list(**void**) {**return** pfaces\_list\_;}
134. num\_of\_vertex\_list(**void**) {**return** pvertices\_list\_ ? **static\_cast**<**int**>(pvertices\_list\_->size()) : 0;}
135. num\_of\_half\_edges\_list(**void**) {**return** pedges\_list\_ ? **static\_cast**<**int**>(pedges\_list\_->size()) : 0;}
136. num\_of\_edge\_list(**void**) {**return** num\_of\_half\_edges\_list()/2;}
137. num\_of\_face\_list(**void**) {**return** pfaces\_list\_ ? **static\_cast**<**int**>(pfaces\_list\_->size()) : 0;}
138. num\_of\_components(**void**) {**return** num\_components\_;}
139. average\_edge\_length(**void**) {**return** average\_edge\_length\_;}
140. get\_vertex(**int** id) {**return** id >= num\_of\_vertex\_list() || id<0 ? NULL : (\*pvertices\_list\_)[id];}
141. get\_half\_edge(**int** id) {**return** id >= num\_of\_half\_edges\_list() || id<0 ? NULL : (\*pedges\_list\_)[id];}
142. get\_face(**int** id) {**return** id >= num\_of\_face\_list() || id<0 ? NULL : (\*pfaces\_list\_)[id];}
143. get\_edge(HE\_vert\* hv0, HE\_vert\* hv1)
144. getWrongNorFacets() { **return** wro\_Nor\_facets\_; }
145. isValid(**void**)
146. InsertVertex(**const** Vec3f& v);
147. InsertVertex( HE\_vert\* & v);
148. InsertEdge(HE\_vert\* vstart, HE\_vert\* vend);
149. InsertFace(std::vector<HE\_vert\* >& vec\_hv);
150. LoadFromOBJFile(**const** **char**\* fins);
151. WriteToOBJFile(**const** **char**\* fouts);
152. LoadFromSTLFile(**const** **char** \* fins);
153. UpdateMesh(**void**);
154. UpdateNormal(**void**);
155. ComputeBoundingBox(**void**);
156. get\_face(**int** vId0, **int** vId1, **int** vId2);
157. get\_face(**const** std::vector<unsigned **int**>& ids);
158. GetFaceId(HE\_face\* face);
159. ResetVertexSelectedTags(**int** tag=UNSELECTED);
160. ResetFaceSelectedTags(**int** tag=UNSELECTED);
161. isNeighbors(HE\_vert\* v0, HE\_vert\* v1);
162. GetSelectedVrtId();
163. CreateMesh(**const** std::vector<Vec3f>& verts, **const** std::vector<**int**>& triIdx);
164. CreateMesh(**const** std::vector<**double**>& verts, **const** std::vector<unsigned>& triIdx);
165. GetBoundaryVrtSize();
166. meshTranslate(**float** param1, **float** param2);
167. **public**:
168. ClearData(**void**);
169. **private**:
170. ClearVertex(**void**);
171. ClearEdges(**void**);
172. ClearFaces(**void**);
173. ComputeFaceslistNormal(**void**);
174. ComputePerFaceNormal(HE\_face\* hf);
175. ComputeVertexlistNormal(**void**);
176. ComputePerVertexNormal(HE\_vert\* hv);
177. ComputeAvarageEdgeLength(**void**);
178. SetBoundaryFlag(**void**);
179. BoundaryCheck();
180. countBoundaryComponat();
181. Unify(**float** size);
182. isFaceContainVertex(HE\_face\* face, HE\_vert\* vert);
183. get\_neighborId(**const** **size\_t**& vertid, std::vector<**size\_t**>& neighbors)
184. **private**:
185. SetNeighbors()
186. **public**:
187. LinearTex()
188. SphereTex()
189. getBoundingBox()
190. getedgemap()
191. Transformation(**float** \* matrix);
192. SetDirection(**int** faceid);
193. **private**:
194. ClearSlice();
195. **public**:
196. InsertFaceSup(std::vector<HE\_vert\* >& vec\_hv);
197. UpdateMeshSup(**void**);
198. GetBLoop() { **return** bLoop; }
199. UpdateBList(**void**);
200. computeComponent();
201. FaceDFS(HE\_face\* facet, **int** no);
202. GetBhelist() { **return** bheList; }

##### 源程序文件说明

RenderingWidget.h 和RenderingWidget.cpp是Meshprint系统对各模块功能实现的接口文件，例如void RenderingWidget::ReadMesh()函数就是读取模型的接口。在该函数中通过调用Mesh3D类中相应的函数（如 bool LoadFromOBJFile(const char\* fins)或者bool LoadFromSTLFile(const char \* fins)）来实现读取模块的功能。因此，模块——读取模型，用到的文件有：RenderingWidget.h 、RenderingWidget.cpp、Mesh3D.h和Mesh3D.cpp。Mesh3D.h和Mesh3D.cpp在HE\_mesh文件夹中。

##### 函数说明

（1）该模块用到的主要函数有：

1. **void** RenderingWidget::ReadMesh()
2. **bool** Mesh3D::LoadFromOBJFile(**const** **char**\* fins)
3. **bool** Mesh3D::LoadFromSTLFile(**const** **char**\* fins)

（2）函数1在文件RenderingWidget.cpp实现，函数2和函数3在Mesh3D.cpp中实现。在函数**void** RenderingWidget::ReadMesh()判断读取文件的后缀后，根据读取文件的后缀决定调用对应的函数2或者函数3。函数2和函数3的参数都为文件所在的路径。

1. **if** (fileinfo.suffix() == "obj")
2. ptr\_mesh\_->LoadFromOBJFile(byfilename.data());
3. **else** **if** (fileinfo.suffix() == "stl" || fileinfo.suffix() == "STL")
4. ptr\_mesh\_->LoadFromSTLFile(byfilename.data());

（3）函数**bool** Mesh3D::LoadFromOBJFile(**const** **char**\* fins)主要由两部分组成，读取vertex信息，和读取face信息。其中主要靠下列三个函数将读取的信息存储到相应的链表中。

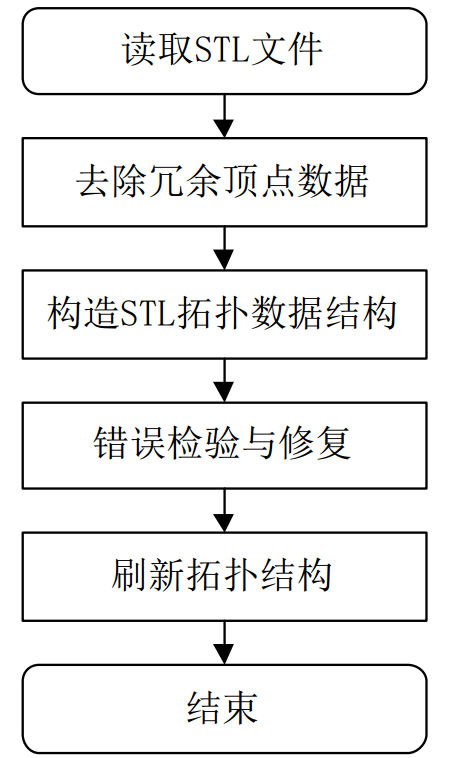
1. HE\_vert\* Mesh3D::InsertVertex(**const** Vec3f& v)
2. HE\_edge\* Mesh3D::InsertEdge(HE\_vert\* vstart, HE\_vert\* vend)
3. HE\_face\* Mesh3D::InsertFace(std::vector<HE\_vert\* >& vec\_hv)

##### 限制条件

##### 其他说明

#### 模型修复子模块

##### 设计图



##### 功能描述

在分析STL模型常见错误的基础上，本项目进行深入研究，主要工作流程如图2.5所示。针对读入的 STL 模型，首先去处 STL 文件中的冗余信息，同时建立高效合理的拓扑数据结构。在此基础上检验 STL 文件模型存在的错误，并给出相应的错误修复算法，完成了文件自动修复功能，提高了系统的实用性和稳定性。

##### 输入数据

STL是最多[快速原型](http://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E5%8E%9F%E5%9E%8B)系统所应用的标准文件[类型](http://baike.baidu.com/item/%E7%B1%BB%E5%9E%8B/33084)。STL是用三角网格来表现3D CAD模型。在STL文件中的三角面片的信息单元 facet 是一个带矢量方向的三角面片，STL三维模型就是由一系列这样的三角面片构成。

整个STL文件的首行给出了文件路径及文件名。

在一个 STL文件中，每一个facet由7 行数据组成，

facet normal 是三角面片指向实体外部的法矢量坐标，

outer loop 说明随后的3行数据分别是三角面片的3个顶点坐标，3顶点沿指向实体外部的法矢量方向逆时针排列。[1]

ASCII码格式的STL文件逐行给出三角面片的几何信息，每一行以1个或2个关键字开头。ASCII格式的STL 文件结构如下：

1. 明码://字符段意义
2. solidfilenamestl//文件路径及文件名
3. facetnormalxyz//三角面片法向量的3个分量值
4. outerloop
5. vertexxyz//三角面片第一个顶点坐标
6. vertexxyz//三角面片第二个顶点坐标
7. vertexxyz//三角面片第三个顶点坐标
8. endloop
9. endfacet//完成一个三角面片定义
11. ......//其他facet
13. endsolidfilenamestl//整个STL文件定义结束

二进制STL文件用固定的字节数来给出三角面片的几何信息。文件起始的80个字节是文件头，用于存贮文件名；紧接着用 4 个字节的整数来描述模型的三角面片个数，后面逐个给出每个三角面片的几何信息。每个三角面片占用固定的50个字节，依次是:

3个4字节浮点数(角面片的法矢量)

3个4字节浮点数(1个顶点的坐标)

3个4字节浮点数(2个顶点的坐标)

3个4字节浮点数(3个顶点的坐标)个

三角面片的最后2个字节用来描述三角面片的属性信息。

一个完整二进制STL文件的大小为三角形面片数乘以 50再加上84个字节。

二进制格式的STL 文件结构如下:

1. UINT8//Header//文件头
2. **UINT32**//Numberoftriangles//三角面片数量
3. //foreachtriangle（每个三角面片中）
4. REAL32[3]//Normalvector//法线矢量
5. REAL32[3]//Vertex1//顶点1坐标
6. REAL32[3]//Vertex2//顶点2坐标
7. REAL32[3]//Vertex3//顶点3坐标
8. UINT16//Attributebytecountend//文件属性统计

##### 输出数据

该子模块输出的数据正是后续模块操作的输入，即模型点线面的三维拓扑结构信息，是以半边数据结构形式存储在内存中的数据。

##### 算法和流程

（1）法线方向错误修复：当根据其中的一条边检验出这种错误时， 通过右手定则， 把有错误方向的三角形矢量各个坐标的值分别改变正负符号即可。

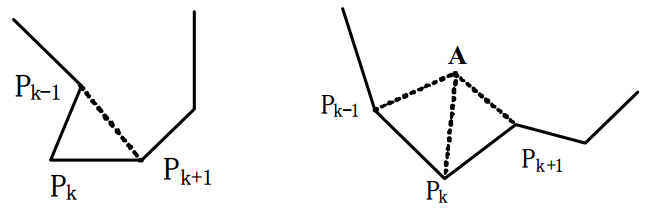
（2）孔洞修复：若模型中某条边仅属于一个三角形时，则出现孔洞。我们借鉴张丽艳的算法进行孔洞修复。即通过建立由孔洞边界组成的空间孔洞多边形特征面，把对空间孔洞多边形的三角形划分问题投影到平面上进行处理。然后根据投影后的孔洞多边形边界的夹角关系不断生成新增的三角片来填充孔洞。这样被填充区域的三角片形状较为优化，避免了狭长及错误三角片的出现，而且新产生的修补三角片的形状及密度与孔洞周围原有三角形网格相协调。具体算法步骤如下：

a) 孔洞多边形特征面的确定。 通过错误检验，获得孔洞的边界之后，对一个单个孔洞Hi，计算Hi边界上的所有边界边的顶点的形心Oi，作为该Hi的特征面的原点，用主元素分析法计算特征面的法矢Ni，从而得到孔洞多边形的特征面Pi。由Oi和Ni，可以 建立一个局部坐标系，把Hi的边界边投影到特征面上成为特征多边形，并将特征多边形坐标转化为局部坐标，此时就把孔洞填充转化到对该特征多边形的三角划分上来。

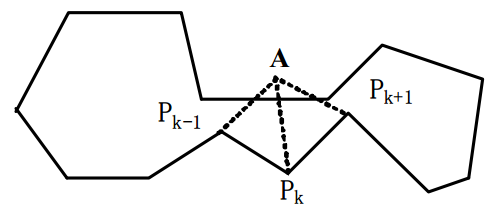
b) 构造新三角片。每次寻找特征多边形中夹角最小的一对邻边，根据这对邻 边的夹角大小（分为大于90 度和小于90 度两种）采用两种不同的方式生成新的三角片(如图所示)，然后更新该特征多边形，再次对夹角最小的一对邻边生 成三角片，直至新增三角片覆盖整个孔洞。在具体实现的时候需要注意新增三角片 的方向必须同原始模型中三角片的方向保持一致。

c) 新增三角片合法性检查。每生成一个新增三角片，都应该检查该三角片的合法性，防止产生相交三角片。（图3.9所示）

具体实现时，对每个孔洞边界建立记录一个边界边的有序链表，保证链表表示的边界边按照构成孔洞的顺序排列，所有孔洞边界的链表头部可记录在一个一维数组中。



新三角面片生成



孔洞多边形填充

图2.9 孔洞多边形填充

顶点错误修复

上述 STL 模型修复功能基本实现了主要的文件错误的修复，在模型读取的时候直接进行模型检验与修复。

##### 源程序文件说明

RenderingWidget.h 和RenderingWidget.cpp是Meshprint系统对各模块功能实现的接口文件，例如void RenderingWidget::ReadMesh()函数就是读取模型的接口。在该函数中通过调用Mesh3D类中的函数bool LoadFromSTLFile(const char \* fins)）来实现读取模块的功能。

##### 函数说明

（1）对于法向错误的面片，改变三角面片中三条线的指向：

1. **if** (he1->pface\_  != NULL  || he2->pface\_  != NULL  || he3->pface\_  != NULL)//if has one been used, mean the triangle is stored in cw direction.
2. {
3. qDebug() << "insert #" << (pfaces\_list\_->size()) << "facet,he1 has been used in facet:";
4. pface->set\_selected(SELECTED);
5. sss++;
6. pface->pedge\_  = he1\_pair\_;
7. he1\_pair\_->pnext\_  = he3\_pair\_;
8. he1\_pair\_->pprev\_  = he2\_pair\_;
9. he1\_pair\_->pface\_  = pface;
10. he2\_pair\_->pnext\_  = he1\_pair\_;
11. he2\_pair\_->pprev\_  = he3\_pair\_;
12. he2\_pair\_->pface\_  = pface;
13. he3\_pair\_->pnext\_  = he2\_pair\_;
14. he3\_pair\_->pprev\_  = he1\_pair\_;
15. he3\_pair\_->pface\_  = pface;
16. HE\_edge \*current  = pface->pedge\_->pnext\_->pnext\_;
17. vec\_hv[0]->adjHEdges.push\_back(he1);
18. vec\_hv[1]->adjHEdges.push\_back(he2);
19. vec\_hv[2]->adjHEdges.push\_back(he3);
20. }

（2）对模型空洞的查找在countBoundaryComponat() 中实现。

1. **void** Mesh3D::countBoundaryComponat()
2. {
3. **if** (bLoop.size()!=0)
4. {
5. **return**;
6. bLoop.clear();
7. }
8. **else**
9. {
10. bLoop.resize(no\_loop + 1);//initial the vectro bloop
11. }
12. //count the number of boundary loops
13. **size\_t** i;
14. **for** (i = 0; i < bheList->size(); i++)
15. {
16. HE\_edge \*cur = bheList->at(i);
17. HE\_edge \*nex = cur;
18. **while** (nex->start\_->selected() != SELECTED)
19. {
20. bLoop[no\_loop].push\_back(nex);
21. nex->start\_->set\_seleted(SELECTED);
22. nex = nex->pvert\_->pedge\_;
23. **if** (nex == cur)
24. {
25. no\_loop++;
26. bLoop.resize(no\_loop+1);
27. **break**;
28. }
29. }
30. }
31. **for** (i = 0; i < bheList->size(); i++)
32. {
33. bheList->at(i)->start\_->set\_seleted(UNSELECTED);
34. }
35. bLoop.resize(no\_loop);
36. bLoop;
37. }

##### 限制条件

##### 其他说明

### 添加支撑模块

添加支撑模块主要包含：手动添加点支撑、手动添加线支撑、自动生成线支撑。

#### 手动添加点支撑子模块

##### 功能描述

该子模块可以实现通过手动点选模型表面来在点选位置添加点支撑，点支撑的长宽参数可以在该模块更改。

##### 输入数据

该模块的输入是鼠标在窗口中的像素值。在程序进入添加点支撑模式时，点击鼠标左键即将光标所在窗口中的像素点坐标输入给了该模块。

##### 输出数据

添加支撑模块利用上一小节讲到的像素点，找到在世界坐标系中，模型实际对应的三维笛卡尔坐标值。以该点出发向下生成包含点线面信息的三维实体点支撑结构，该结构即为此次添加点支撑的输出数据。

##### 业务算法和流程

1. 对要处理的模型，用Octree数据结构将其面片信息存储到相应的节点中。
2. 通过反投影方法找到通过当前像素点垂直于屏幕的直线上的两个点，他们分别是在屏幕外和屏幕里的两个世界坐标系中的两个点。
3. 连接两点生成射线，用射线求在世界坐标系中其与模型的交点，这一求交点的过程需要用到第一步建立起来的Octree，如此可以在O（lgn）的时间内求得交点。
4. 在上一步结束后，对得到的交点向z轴的反方向做射线并求其与模型的交点，该过程同步骤（3）。
5. 生成四棱柱支撑。

##### 数据设计

该模块处用到了八叉树（Octree）数据结构，八叉树是一种用于描述三维空间的树状数据结构。八叉树的每个节点表示一个正方体的体积元素，每个节点有八个子节点，将八个子节点所表示的体积元素加在一起就等于父节点的体积。八叉树就是用在3D空间中的场景管理，可以很快地知道物体在3D场景中的位置，或侦测与其它物体是否有碰撞以及是否在可视范围内。

本例中八叉树的存贮结构用一个有（若干+八）个字段的记录来表示树中的每个结点。其中若干字段用来描述该结点的特性(本例中的特性为：节点的值和节点坐标)，其余的八个字段用来作为存放指向其八个子结点的指针。此外，还有线性存储和1托8式存储。

1. AlignAxisBoundingBox
2. **public**:
3. max\_point;
4. min\_point;
5. AlignAxisBoundingBox(Vec3f max\_p, Vec3f min\_p)
6. AlignAxisBoundingBox(**void**) {}
7. AABB;
8. MeshOcNode
9. **public**:
10. MeshOcNode()
11. ~MeshOcNode()
12. child\_node\_[8];
13. ocn\_aabb\_;
14. ocn\_face\_list\_;
15. is\_leaf\_;
16. MeshOctree
17. **public**:
18. oc\_root\_;
19. oc\_face\_list\_;
20. bary\_center;
21. MeshOctree() : oc\_root\_(NULL) , oc\_face\_list\_(NULL)
22. ~MeshOctree() { DeleteMeshOcTree(); }
23. DeleteMeshOcTree();
24. BuildOctree(Mesh3D\* ptr\_in);
25. InteractPoint(Vec3f point\_in, Vec3f d, **bool** hitSelf = **false**, **const** std::vector<**int**>& faceRegionLabel = std::vector<**int**>(), **int** curRegion = 0);
26. hitOctreeNode(MeshOcNode\* node\_in, Vec3f point\_in, Vec3f& point\_out, Vec3f d, **int** &hitID, **float** &t, **bool** hitSelf = **false**, **const** std::vector<**int**>& faceRegionLabel = std::vector<**int**>(), **int** curRegion = 0);
27. **private**:
28. createOctree(MeshOcNode\*& node\_in, std::vector<**int**>\* face\_list\_idx, AABB aabb, **bool** stopFlag);
29. MergeBoundingBox(AABB &A, AABB A1, AABB A2)
30. RayHitAABB(Vec3f raySPoint, Vec3f rayDirection, Vec3f A, Vec3f B);
31. isHitTriangle(HE\_face\* face\_in, Vec3f point\_in, Vec3f& point\_out, Vec3f d, **bool** hitSelf);
32. isCollidAABB(AABB ab\_1, AABB ab\_2)

##### 源程序文件说明

Support.h和Support.cpp两个文件实现了支撑的所有功能。

##### 函数说明

（1）生成点支撑的入口函数是：

1. **void** RenderingWidget::mousePressEvent(QMouseEvent \*e)

（2）局部变量有

1. Vec3f myPointN(e->x(), height() - e->y(), -1.0f);
2. Vec3f myPointF(e->x(), height() - e->y(), 1.0f);
3. Vec3f add\_pointN = myProjector.UnProject(myPointN);
4. Vec3f add\_pointF = myProjector.UnProject(myPointF);
5. Vec3f direc = add\_pointF - add\_pointN;

（3）找到交点的函数为：

1. **void** MeshOctree::hitOctreeNode(MeshOcNode\* node\_in, Vec3f point\_in, Vec3f& point\_out, Vec3f d, **int** &hitID, **float** &t, **bool** hitSelf, **const** std::vector<**int**>& faceRegionLabel, **int** curRegion)

（4）生成支撑的函数为：

1. **void** Support::AddPointSupport(Vec3f add\_point, **float** p\_width)

（5）最后更新Mesh的点边面拓扑结构以及Octree的结构的函数分别为：

1. **inline** **void** updateSupportMesh() { mesh\_support\_->UpdateMesh(); }
2. **void** BuildSupportOctree()

##### 限制条件

##### 其他说明

#### 手动添加线支撑子模块

##### 设计图

##### 功能描述

该子模块可以实现通过手动点选模型表面来在两个点选位置添加线支撑，线支撑的宽度和凸台高度参数可以在该模块更改。

##### 输入数据

该模块的输入是鼠标在窗口中的像素值。在程序进入添加线支撑模式时，连续点击鼠标左键两次，即将光标所在窗口中的像素点坐标输入给了该模块。

##### 输出数据

添加支撑模块利用上一小节讲到的像素点，找到在世界坐标系中，模型实际对应的三维笛卡尔坐标值。以该点出发向下生成包含点线面信息的三维实体线支撑结构，该结构即为此次添加线支撑的输出数据。

##### 业务算法和流程

1. 对要处理的模型，用Octree数据结构将其面片信息存储到相应的节点中。
2. 通过反投影方法找到通过当前像素点垂直于屏幕的直线上的两个点，他们分别是在屏幕外和屏幕里的两个世界坐标系中的两个点。将该点存到数组line\_points\_中，如果line\_points\_的个数小于2，返回，否则继续（3）（4）（5）步。
3. 在两点之间沿其两点的连线方向等间距的取一系列的点，将这些点存到数组toAdd中。
4. 在上一步结束后，对得到的toAdd的所有点点向z轴的反方向做射线并求其与模型的交点，该过程同步骤（3）。
5. 根据所求得的交点信息生成线支撑。

##### 数据设计

此处的数据结构设计完全等同于上一子模块。

##### 源程序文件说明

涉及到的程序文件共有renderingwidget.h、renderingwidget.cpp、support.h和support.cpp四个文件，都在程序安装包的文件夹之内。

##### 函数说明

（1）生成线支撑的入口函数是：

1. **void** RenderingWidget::mousePressEvent(QMouseEvent \*e)

（2）局部变量有

1. Vec3f myPointN(e->x(), height() - e->y(), -1.0f);
2. Vec3f myPointF(e->x(), height() - e->y(), 1.0f);
3. Vec3f add\_pointN = myProjector.UnProject(myPointN);
4. Vec3f add\_pointF = myProjector.UnProject(myPointF);
5. Vec3f direc = add\_pointF - add\_pointN;

（3）找到交点的函数为：

1. **void** MeshOctree::hitOctreeNode(MeshOcNode\* node\_in, Vec3f point\_in, Vec3f& point\_out, Vec3f d, **int** &hitID, **float** &t, **bool** hitSelf, **const** std::vector<**int**>& faceRegionLabel, **int** curRegion)

（4）生成支撑的函数为：

1. **void** Support:: AddLineSupport (Vec3f add\_point, **float** p\_width)

（5）最后更新Mesh的点边面拓扑结构以及Octree的结构的函数分别为：

1. **inline** **void** updateSupportMesh() { mesh\_support\_->UpdateMesh(); }
2. **void** BuildSupportOctree()

##### 限制条件

##### 其他说明

#### 自动添加线支撑子模块

##### 功能描述

自动找出与Z轴负方向夹角过大，需要被支撑的所有面片。并且，将相互紧邻的面片合并到同一区域，探测出每个区域的边界线，边界线作为该区域的围墙，以线支撑的形式将区域包围起来，并在围墙区域内部添加等间隔等间距的线支撑。

##### 输入数据

输入数据即为读取的模型。

##### 输出数据

生成支撑结构，并将支撑结构的点线面存储成Mesh3D的一个对象，即满足半边数据结构的三维模型。

##### 业务算法和流程

（1）遍历并标记所有需要支撑的面片，我们假设所有法向与z轴负方向夹角小于θ的面片都需要支撑；

（2）使用区域增长算法，对所有连通的区域进行合并，统计出区域的个数；

（3）在每个连通区域内找出所有边界边的信息，并对其内部进行填充（调用手动支撑部分的函数），其中进行填充的内部参数尽量可调，如填充密集程度，填充部分分辨率，以及填充杆的格式等。

##### 数据设计

数据设计同其他子模块。

##### 源程序文件说明

涉及到的程序文件共有renderingwidget.h、renderingwidget.cpp、support.h和support.cpp四个文件，都在程序安装包的文件夹之内。

##### 函数说明

（1）生成自动支撑的入口函数是：

1. **void** RenderingWidget::AutoSupport()

（2）局部变量有

1. std::vector<**bool**> faceNeedSupportFlag(faceNum);
2. std::vector<**int**> neighorNeedSupportNum(faceNum);
3. std::vector<**int**>\* boundaryEdges = **new** std::vector<**int**>[regionNum]; // boundaryEdges that belong to the region
4. std::vector<**int**>\* boundaryEdgesSep = **new** std::vector<**int**>[regionNum]; // separate different boundaries (when a region has more than one boundaries)
5. std::vector<**bool**>\* boundaryOuterLoopFlag = **new** std::vector<**bool**>[regionNum]; // inner-boundary or outer-boundary
6. std::vector<**float**>\* boundaryEdgesLeftMostX = **new** std::vector<**float**>[regionNum]; // sth similiar to AABB, help assign values to boundaryOuterLoopFlag

（3）对边界向内偏置一个距离后，利用下面的函数生成围墙：

1. std::vector<Vec3f> Support::AddLineSupport(std::vector<Vec3f> add\_points, **bool** isOuter, **const** std::vector<**int**>& regions, **int** curRegion)

（4）在边界内生成内部线支撑的样点：

1. **void** RayIntersection2d(Vec3f sPoint, Tree2dNode \*node, std::vector<LineSegment\*> segments, std::vector<Vec3f> &hitPointList)

（5）最后更新Mesh的点边面拓扑结构以及Octree的结构的函数分别为：

1. **inline** **void** updateSupportMesh() { mesh\_support\_->UpdateMesh(); }
2. **void** BuildSupportOctree()

##### 限制条件

##### 其他说明

### 生成路径模块

添加支撑模块主要包含：切层模块、路径规划模块以及AFF文件生成模块。

#### 模型切层子模块

##### 功能描述

依据STL模型的几何连续特征，研究基于STL模型几何连续性的分层切片算法，即通过对构成模型的所有三角面片分组，构建求交链表，在求交链表内部建立面片间的局部拓扑关系，进而完成切平面与三角面片的求交运算，求交后将得到的交点按照一定顺序连接，从而的到封闭的轮廓信息，最后查找并删除冗余点。

##### 输入数据

输入数据为模型的网格结构和支撑的网格数据，分别为Mesh3D \*ptr\_mesh\_和Mesh3D \*mesh\_support\_。

##### 输出数据

对于每一个切层厚度，该子模块可以生成若干个闭合的轮廓，轮廓有若干个前后收尾相连的边构成，并且这些边以逆时针顺序存放在链表中。SliceCut \*mycut和SliceCut \*mycutsup分别存放模型的轮廓和支撑的轮廓。

##### 业务算法和流程

（1）分组：对于任一个三角面片找出其三个点的最小Z值Zmin，模型

分层起始高度为Z0 ，与与该三角面片最先相交的层面序号i=[(Zmin-Z0)/ △Z]，i取整，计算后将该三角面片加入层号为i的集合中；

（2）构造求交链表：构造一个单向链表来存储参与求交计算的所有三

角面片，对于求交链表中的任一个三角面片，找出最大的Zmax,并与当前切平面高度Zi进行比较，若Zmax<Zi,说明该三角面片完全处于当前切平面的下方，不想交，则将该三角面片从求交链表中删除，直到将所有与Zi切平面不相交的三角面片从求交链表中全部删除，之后从分组矩阵中提出去与Zi切平面新相交的三角面片插入链表中，进行新一轮的求交运算。

##### 数据设计

此部分用到了两个类，一个用来表示轮廓线段的cutLine，一个是SliceCut类。两个类的代码都在slice.h和slice.cpp中实现。cutLine主要的成员变量为position\_vert[2]——存储线段两个端点的位置坐标。而SliceCut类的变量——pieces\_list\_是一个存储所有层所有轮廓边的链表，storage\_Face\_list\_是切层过程中用到的变量，是每一层存放面片地址的链表。

1. cutLine
2. **public**:
3. cutLine() {}
4. cutLine(point p1, point p2)
5. cutLine(point p1, point p2,**int** xID,**int** yID)
6. ~cutLine()
7. position\_vert[2];
8. edgeid\_vert[2];
9. SliceCut
10. **public**:
11. SliceCut()
12. SliceCut(Mesh3D\* ptr\_in, **float** tn = DEFAULT\_T) :mesh\_in\_(ptr\_in)
13. ~SliceCut();
14. SetThickness(**float** tn = DEFAULT\_T)
15. clearcut();
16. CutInPieces();
17. Exportslice();
18. getThickness() { **return** thickness\_; }
19. GetPieces(){ **return** pieces\_list\_; }
20. GetNumPieces() { **return** num\_pieces\_; }
21. storeMeshIntoSlice();
22. storage\_Face\_list\_;
23. num\_pieces\_;
24. **private**:
25. mesh\_in\_;
26. circle\_list\_;
27. pieces\_list\_;
28. isEdgeInFace(HE\_vert\* pvert1, HE\_vert\* pvert2, HE\_face\* pface);
29. sortVertInFace(**int** faceid);

##### 源程序文件说明

slice.h和slice.cpp都存放在源程序代码的根目录中。

##### 函数说明

（1）模型生成切层和支撑生成切层的入口函数是：

1. **void** RenderingWidget::cutinPieces()
2. **void** RenderingWidget::cutinPiecesSup()

（2）每个入口函数都会构造一个新的Slicecut对象，通过对象调用其子函数的形式生成轮廓边。其中storeMeshIntoSlice()实现了将三角面片与分层，CutInPieces()实现求交点的运算。

1. mycut = **new** SliceCut(ptr\_mesh\_);
2. mycut->storeMeshIntoSlice();
3. mycut->CutInPieces();

##### 限制条件

##### 其他说明

#### 路径生成子模块

##### 功能描述

在分层得到的轮廓的基础上进行合理的规划和设计，生成各种快速成形工艺的不同扫描路径。

##### 输入数据

路径规划的输入是上一子模块的输出，即每一层的所有有序封闭轮廓线段。

##### 输出数据

路径规划的输出是激光扫描的实际路径，在本程序中，有向线段用其前后两个端点表征，若干条这样的有向线段有序的存放在链表中。

##### 业务算法和流程

1. 计算得到每个黑格中路径线的x值坐标，通过该x值作一条垂直于x轴的直线（在程序中以该x值表示这条直线）；计算每个白格中路径线的y值坐标，通过该y值作一条垂直于x轴的直线（在程序中以该y值表示这条直线。
2. 将该层中所有点顺时针旋转一个角度θ。
3. 对每一层中的每一条闭合轮廓线，从第一条边开始，计算其与步骤（1）中的直线相交点。
4. 将所有线段的端点逆时针旋转角度θ；
5. 对每一条直线，截取其存在于棋盘格中的线段存放到路径中。

##### 数据设计

1. **class** BField
2. {
3. **public**:
4. BField(**int** x, **int** y, **float** z)
5. ~BField()
6. std::vector<Vec3f\*>   hatch\_line\_;
7. **float** leftcoor\_, rightcoot\_, topcoor\_, bottomcoor\_;
8. **private**:
9. **float** topcoordinate\_;
10. **public**:
11. **int** x\_min\_field\_;
12. **int** y\_min\_field\_;
13. **int** x\_max\_field\_;
14. **int** y\_max\_field\_;
15. **float**   z\_height\_;
16. FieldType type\_;
17. FieldType getFieldType() { **return** type\_; };
18. };

在路径生成中，用到了表示棋盘格的类——BField，每个BField 对象都一个表示方格类型的变量type\_，且每个方格都有一个存储其内部路径线的变量hatch\_line\_。

##### 源程序文件说明

该子模块代码全部在hatch.cpp内实现。

##### 函数说明

（1）模型生成路径和支撑生成路径的入口函数是：

1. **void** RenderingWidget::renderdoHatch()

（2）通过对对象的子函数void Hatch::doHatch()调用实现路径规划。进入该函数后首先偏置个一个距离：

1. **for** (**size\_t** j = 0; j < boudary\_edge\_[i].size(); j++)
2. {
3. std::vector<Vec3f> Bpoints;
4. std::vector<std::vector<Vec3f>> OFFPoints;
5. **for** (**size\_t** k = 0; k < (boudary\_edge\_[i])[j]->size(); k++)
6. Bpoints.push\_back((boudary\_edge\_[i])[j]->at(k).position\_vert[0]);
7. Offset(Bpoints, OFFPoints, offset\_dis\_, i);
8. offset\_vert\_[i].insert(offset\_vert\_[i].end(), OFFPoints.begin(), OFFPoints.end());
9. }

（3）对偏置后点所有点旋转，然后计算每个方格的边界点。

1. //for every field, compute hatch line coordinate
2. **float** mid\_hat\_coor\_ = (x\_id\_ + 0.5)\*field\_width\_;
3. **for** (**int** id\_hatch\_ = 0; id\_hatch\_\*line\_width\_ < field\_width\_ / 2; id\_hatch\_++)
4. {
5. **float** x\_coor\_ = id\_hatch\_\*line\_width\_ + mid\_hat\_coor\_;
6. **if** (x\_coor\_ < x\_max\_&&x\_coor\_ >= x\_min\_)
7. {
8. Vec3f point\_p\_ = a + (id\_hatch\_\*line\_width\_ + mid\_hat\_coor\_ - a.x()) / vector\_.x()\*vector\_;
9. hatch\_point\_black\_.insert(point\_p\_);
10. }
11. x\_coor\_ = -id\_hatch\_\*line\_width\_ + mid\_hat\_coor\_;
12. **if** (x\_coor\_ < x\_max\_&&x\_coor\_ >= x\_min\_)
13. {
14. Vec3f point\_n\_ = a + (-id\_hatch\_\*line\_width\_ + mid\_hat\_coor\_ - a.x()) / vector\_.x()\*vector\_;
15. hatch\_point\_black\_.insert(point\_n\_);
16. }
17. }

（4）对每个方格填充其内部路径线：

1. BField \* big\_tempDw\_ = **new** BField(x\_id\_, last\_id\_, k\*thickness\_);//new a big grid only has x and y information
2. std::set<BField\*, compareBField>::iterator iter1 = big\_fields\_.find(big\_tempDw\_);
3. **if** (iter1 == big\_fields\_.end())//if has not found a big grid
4. {
5. **if** (boundaryPD\_.y() - big\_tempDw\_->bottomcoor\_ > 1e-6&&boundaryPD\_.y() - big\_tempDw\_->topcoor\_ < -1e-6)
6. big\_tempDw\_->Black\_hatch\_point\_.insert(boundaryPD\_);
7. big\_fields\_.insert(big\_tempDw\_);
8. }
9. **else**
10. {
11. **if** (boundaryPD\_.y() - big\_tempDw\_->bottomcoor\_ > 1e-6&&boundaryPD\_.y() - big\_tempDw\_->topcoor\_ < -1e-6)
12. (\*iter1)->Black\_hatch\_point\_.insert(boundaryPD\_);
13. **delete** big\_tempDw\_;
14. }

##### 限制条件

##### 其他说明

#### 生成AFF文件子模块

##### 功能描述

上位机生成的路径存放在内存中，并不能直接用于生产，需要导出为下位机可以识别的.AFF格式文件。在此子模块中，根据AFF文件对路径线、激光参数的定义等相关格式，将内存中的路径生成为存放在硬盘里的，后缀为aff的文件。

##### 输入数据

输入数据为存放模型内部路径，模型外围轮廓路径，支撑内部路径以及支撑外围轮廓路径点的数组。

##### 输出数据

输出成具有AFF要求特定格式的二进制文件。

##### 业务算法和流程

1. 在文件中写入头信息，包括文件名称、公司名称、模型坐标范围、层厚等文件头信息。
2. 对每一层每一个联通区域，计算其扫描路径所需要占用的字节数C，记录C到文件中，随后依次将路径各点的坐标写到文件中。
3. 结束。

##### 源程序文件说明

生成AFF文件的功能在renderingwidget.cpp文件中。函数名为void RenderingWidget::Export()。

##### 函数说明

（1）函数中用到的局部变量如下表，其中1-10行依次为层数、激光功率、模型尺寸范围、激光速度、层厚。15-24行为常量，是AFF格式所要求的标签。

1. **int** layers = mycut->GetNumPieces() - 1;
2. **float** power = myhatch->getLaserPower();
3. **float** xmin = ptr\_mesh\_->getBoundingBox().at(1).x();
4. **float** xmax = ptr\_mesh\_->getBoundingBox().at(0).x();
5. **float** ymin = ptr\_mesh\_->getBoundingBox().at(1).y();
6. **float** ymax = ptr\_mesh\_->getBoundingBox().at(0).y();
7. **int** zmin = 300;
8. **int** zmax = (myhatch->GetNumPieces() - 1) \* 300;
9. **float** speed = myhatch->getLaserSpeed();
10. thickness\_ = mycut->getThickness();
11. **int**  THICKNESS = thickness\_ \* 10000;
12. std::vector<Vec3f \*>\* tc = myhatch->getHatch();
13. std::vector < std::vector<cutLine>\* >\*tc2 = (mycut->GetPieces());
14. std::vector < std::vector<Vec3f>>\* tc3 = myhatch->getOffsetVertex();
15. **const** **int** Layer\_Section = 1;
16. **const** **int** LayerZpos\_Section = 12;
17. **const** **int** Polygon\_Section = 2;
18. **const** **int** Hatch\_Section = 4;
19. **const** **int** LaserPower\_Section = 6;
20. **const** **int** LaserSpeed\_Section = 7;
21. **const** **int** FocusShift\_Section = 8;
22. **const** **int** PolygonCoordinates\_Section = 3;
23. **const** **int** HatchCoordinates\_Section = 5;
24. **const** **int** PartID\_Section = 5;

（2）对每一层路径，需要的局部变量有，11-21行为定义的存放相应数据的字节流。

1. **int** zposition = i\*myhatch->getThickness() \* 10000;
2. **int** len\_layer = 0;
3. **int** len\_Zpos = 4;
4. **int** len\_Polygon = 0;
5. **int** len\_Hatch = 0;
6. **int** len\_LaserPower = 4;
7. **int** len\_laserSpeed = 4;
8. **int** len\_Part = 2;
9. **int**\*len\_PolygonCoor = NULL;
10. **int** len\_HatchCoor = 0;
11. QByteArray newlayer = NULL;
12. QByteArray zPosition = NULL;
13. QByteArray hatch = NULL;
14. QByteArray hatchCoor = NULL;
15. QByteArray polygon = NULL;
16. QByteArray\* polygonCoor = NULL;
17. QByteArray laserPower = NULL;
18. QByteArray laserSpeed = NULL;
19. QByteArray laserPowerPolygon = NULL;
20. QByteArray laserSpeedPolygon = NULL;
21. QByteArray partID = NULL;

（3）完成对所有字节流的附着后，将这些QByteArray 写入文件中：

1. newlayer.append(**reinterpret\_cast**<**const** **char** \*>(&Layer\_Section), 2);
2. newlayer.append(**reinterpret\_cast**<**const** **char** \*>(&len\_layer), 4);
3. newlayer.append(zPosition);
4. newlayer.append(hatch);
5. newlayer.append(polygon);
6. outBinary.writeRawData(newlayer, **sizeof**(**char**) \* (newlayer.size()));;

##### 限制条件

##### 其他说明

# 接口设计

## 内部接口

程序的内部接口指实现三个模块功能的接口：

1. **public** slots:
2. **void** ReadMesh();
3. **void** WriteMesh();
4. **void** Export();
5. **void** SelectFace(**int** x, **int** y);
6. **void** FindRegion();
7. **void** cutinPieces();
8. **void** cutinPiecesSup();
9. **void** renderdoHatch();
10. **void** objectTransformation(**float** \* matrix);
11. **void** Translation();

## 外部接口

程序的外部接口如下表，这些接口实现了对各子模块参数的设置、模型在窗口内的显示模式（显示线、面、支撑、支撑区域、灯光效果）以及实现了需要进入的操作模式（包括添加点支撑模式、添加线支撑模式、删除支撑模式、选择打印地面模式）。

1. **void** SetSliceCheckId(**int** id);
2. **void** CheckDrawPoint();
3. **void** CheckDrawEdge();
4. **void** CheckDrawFace();
5. **void** CheckLight();
6. **void** CheckGrid();
7. **void** CheckDrawTexture();
8. **void** CheckDrawAxes();
9. **void** CheckDrawCutPieces();
10. **void** Checkmoduletranslate();
11. **void** CheckSetFace();
12. **void** CheckRegion(**bool** bv);
13. **void** CheckSupport(**bool** bv);
14. // support operators
15. **void** AddPointSupport();
16. **void** AddLineSupport();
17. **void** DeleteSupport();
18. **void** AutoSupport();
19. **void** setPointD(**double** diameter);
20. **void** setPointH(**double** diameter);
21. **void** setLineD(**double** diameter);
22. **void** setLineH(**double** diameter);
23. **void** setThreshold(**double** threshold) { THRESHOLD = cos(3.1415926 \* threshold / 180); };
24. **void** setGap(**double** gap) { GAP = gap; };
25. **void** setSeglength(**double** length) { SEGLENGTH = length; };
26. **void** setReso(**double** reso) { RESO = reso; };
27. **void** setVerticalgap(**double** verticalgap) { VERTICALGAP = verticalgap; };
28. **void** setfieldWidth(**double** width);
29. **void** setfieldHeight(**double** height);
30. **void** setlineOverlap(**int** lineoverlap);
31. **void** setfieldOverlap(**double** fieldoverlap);
32. **void** setThickness(**double** thick);
33. **void** setAngle(**int** angle);
34. **void** setHatchType(**int** type\_)
35. **void** setLaserPower(**int** power) { laser\_power\_hatch\_ = power; };
36. **void** setLaserSpeed(**int** speed) { laser\_speed\_hatch\_ = speed; };
37. **void** setLaserPowerPolygon(**int** power) { laser\_power\_polygon\_ = power; };
38. **void** setLaserSpeedPolygon(**int** speed) { laser\_speed\_polygon\_ = speed; };
39. **void** setThickness(**float** thickness) { thickness\_ = thickness; };
40. **void** SetDirection();

# 数据库设计

此系统的研发过程中未用到数据库。

# 系统安全保密设计

由于不存在与外部系统的接口,所以不需要考虑访问安全的问题

# 系统性能设计

# 系统出错处理