山东大学
SHANDONG UNIVERSITY

毕业论文（设计）

论文（设计）题目：

面向昂贵碳素材料套料加工的交互式布局系统

姓　　名　顾丹禾
学　　号　201900130043
学　　院　计算机科学与技术
专　　业　计算机科学与技术
年　　级　2019 级
指导教师　赵海森

2023 年 5 月 20 日

摘　　要

碳素材料是一种昂贵且先进的工业装备制造用原材料，即高性能的碳纤维材料，广泛应用于航空航天等领域。由于这种材料为工业装备制造所用，是一种消耗品，所以需要满足生产量大的特点。但是碳纤维材料的制造过程非常复杂，需要使用高温高压等特殊工艺，因此成本较高。此外，碳纤维材料的生产过程也会产生大量的废料和污染物，环保要求也增加了成本。套料加工是指可以通过计算机辅助设计软件将多个小型零件的形状进行优化排列，以最大限度地减少材料浪费。碳素材料的零件加工，一般需要经过粗加工的取料阶段，大致可以理解为在一个标准几何体中取出一些特定布局排列的小的几何体，然后再进入数控机床进行更精细的加工制造。这种粗加工的取料问题需要考虑到刀具的可达性约束，即碰撞情况的避免考量。本文拟研究初步的基于交互的，进一步基于几何算法的布局方案，可以应用于工厂的生产制造，旨在提高材料的利用率。

关键字：交互式系统；零件加工；碰撞检测；布局方案

ABSTRACT

Carbon material is a kind of expensive and advanced industrial equipment manufacturing raw material, namely high performance carbon fiber material, widely used in aerospace and other fields. Since this material is used in the manufacture of industrial equipment and is a consumable, it is necessary to meet the characteristics of high production volumes. But carbon fiber is expensive because it is complicated to make and requires special processes such as high temperature and pressure. In addition, the production process of carbon fiber materials also generates a lot of waste and pollutants, and environmental requirements add to the cost. Nesting process refers to the automatic nesting can be carried out by computer aided design software, and the shape of several small parts can be optimized to minimize material waste. Carbon material parts processing, generally need to go through the rough processing of the material stage, roughly can be understood as in a standard geometry to take out some specific layout of the small geometry, and then enter the CNC machine tool for more fine processing and manufacturing. The problem of taking material needs to take into account the accessibility constraints of the tool, that is, the avoidance of collision. This paper intends to study a preliminary layout scheme based on interaction and further based on geometric algorithm, which can be applied to the production and manufacturing of factories in order to improve the utilization rate of materials.

Key Words: Interactive system, Parts processing, Collision detection, Layout scheme

目　录

第 1 章 绪论

1.1 研究背景

碳素材料[1]具有高强度、高刚度、高耐热性等优良性能，被广泛应用于航空等领域。然而，碳素材料的价格昂贵，加工难度大，因此如何减少材料浪费和提高生产效率成为了碳素材料加工领域的研究热点。套料技术[2]是一种有效的解决方案，指可以通过计算机辅助设计软件将多个小型零件的形状进行优化排列，以最大限度地减少材料浪费和提高生产效率。然而，传统的套料技术存在着一些问题，如套料效率低[3]、套料精度不高[4]等。因此，研究面向昂贵碳素材料套料加工的交互式布局系统[5]，具有重要的理论和实际意义。随着人机交互技术[6]的不断发展，交互式设计和优化零件加工的布局成为可能，操作员可以直接参与到零件加工[7]的布局设计和优化中，提高材料的利用率。

本文在碳素材料(图1.1.1)加工企业的某项目基础上，拟研究基于OpenGL的二维图像处理，基于OpenSCAD的三维模型处理，初步的基于交互的，进一步的基于几何算法的零件自动布局方案[8]，旨在提高材料的利用率。

图1.1.1 碳素材料

1.2 国内外研究现状

1.2.1 国内研究现状

国内学者对套料技术的研究相对较少，主要集中在传统的套料算法上。例如，李宏伟等人提出了一种基于贪心算法的套料算法，称为“最大利用率贪心算法[9]”

（Maximum Utilization Greedy Algorithm，MUGA）。该算法的主要思想是在每一步选择最优的方案，以期望达到全局最优解。具体来说，算法的步骤如下：

Ⅰ.将所有待套料的零件按照面积从大到小排序。

Ⅱ.从面积最大的零件开始，依次将其放入可用的空间中，直到无法再放入为止。

Ⅲ.对于剩余的零件，重复步骤2，直到所有零件都被放置完毕。

然而，由于该算法只考虑了当前步骤的最优解，而没有考虑到后续步骤的影响，因此可能会导致局部最优解而非全局最优解。

此外，还有一些学者将遗传算法[10]、模拟退火算法[11]等优化算法应用于套料问题中，取得了一定的成果。遗传算法模拟了生物进化的过程，通过交叉、变异等操作来生成新的解，并通过适应度函数来评估解的质量；而模拟退火算法则是通过模拟物质在高温下的退火过程，来寻找全局最优解。这些算法在套料问题中的应用，可以大大提高套料效率和利用率，减少材料浪费，降低成本。

但是，国内研究还存在着套料效率低、套料精度不高等问题，需要进一步研究和改进。

1.2.2 国外研究现状

国外学者对套料技术进行了广泛的研究。早期的套料算法主要基于贪心算法和回溯算法，但这些算法存在着套料效率低、套料精度不高等问题。近年来，遗传算法、模拟退火算法、禁忌搜索算法[12]等优化算法被广泛应用于套料问题中，取得了较好的效果。例如，K. K. Lai等人提出了一种基于遗传算法的套料算法，该算法的主要思想是通过遗传算法的选择、交叉和变异操作，不断优化套料方案，最终得到最优解。此外，还有一些学者将人工智能技术应用于套料问题中，如神经网络、模糊逻辑等，取得了一定的成果。

1.3 OpenGL[13]和OpenSCAD[14]

1.3.1 OpenGL

1.3.1.1 OpenGL概述

OpenGL是一种跨平台的图形库，它提供了一组函数和数据类型，用于创建和渲染2D和3D图形。在交互式系统的开发中，OpenGL是一个非常重要的工具，可以用于实现各种图形效果和交互功能。

1.3.1.2 OpenGL发展历史

OpenGL的发展历史可以追溯到20世纪80年代初期。当时，计算机图形学的研究者们开始意识到需要一个跨平台的图形API，以便在不同的计算机系统上进行图形编程。于是，他们开始开发OpenGL。

OpenGL最初是由Silicon Graphics公司（SGI）开发的。在1982年，SGI发布了一款名为IRIS 1000的计算机系统，该系统具有强大的图形处理能力。为了充分利用这种能力，SGI开始开发OpenGL。在1987年，SGI发布了OpenGL 1.0版本，这是第一个公开发布的OpenGL版本。

随着时间的推移，OpenGL逐渐成为了计算机图形学领域的标准API。它被广泛应用于游戏开发、虚拟现实、工业设计、科学可视化等领域。同时，OpenGL也不断发展和完善，不断推出新的版本和扩展。例如，OpenGL 2.0引入了着色器程序，OpenGL 3.0引入了统一的着色器语言GLSL，OpenGL 4.0引入了计算着色器等。

除了SGI之外，其他公司和组织也参与了OpenGL的开发和推广。例如，Microsoft开发了DirectX，但它也支持OpenGL。Khronos Group是一个由多家公司组成的非营利组织，它负责OpenGL的维护和推广。目前，OpenGL已经发展到了OpenGL 4.6版本，它仍然是计算机图形学领域的重要API之一。

1.3.1.3 本文涉及到的OpenGL功能

Ⅰ.图形渲染功能

OpenGL最主要的功能之一就是图形渲染。在OpenGL中，我们可以使用各种图形绘制函数来创建各种形状的图形，例如点、线、三角形、矩形等。同时，OpenGL还支持各种材质和纹理，可以让我们的图形更加真实和生动。

在交互式系统中，3D图形是非常常见的。OpenGL提供了各种3D图形的绘制函数和计算方法，可以让我们轻松地创建各种3D图形效果。

Ⅱ.用户交互功能

在交互式系统中，用户与系统的交互是非常重要的。OpenGL提供了各种交互功能，例如鼠标事件、键盘事件、定时器事件等。我们可以通过这些事件来实现各种交互功能，例如拖拽、旋转、缩放等。

Ⅲ.相机控制功能

OpenGL可以通过相机控制来实现视角的变换，例如旋转、平移、缩放等，还可以使用OpenGL的API来控制相机的位置和方向。

1.3.2 OpenSCAD

1.3.2.1 OpenSCAD简介

OpenSCAD是一款自由软件，是一种基于脚本的3D建模工具，它使用一种类似于编程语言的方式来描述3D模型。OpenSCAD的脚本语言非常简单，易于学习和使用，同时也非常强大，可以实现复杂的3D模型设计。OpenSCAD的主要特点如表(1.3.1)所示。

表1.3.1 OpenSCAD的主要特点

特点	内容
基于脚本的3D建模	用户可以通过编写脚本来描述3D模型，而不是通过鼠标和键盘来操作3D模型
参数化设计	用户可以通过定义参数来控制3D模型的各个方面，从而实现快速设计和修改
开放源代码	用户可以自由地使用、修改和分发它
支持多种文件格式	包括STL、OBJ、OFF，可以方便地与其他3D建模软件进行交互

总之，OpenSCAD是一款非常适合程序员和工程师使用的3D建模工具，它可以帮助用户快速地设计和修改复杂的3D模型。

1.3.2.2 本文涉及到的OpenSCAD功能

Ⅰ.基本几何体的创建

OpenSCAD可以创建基本的几何体，如立方体、球体、圆柱体、锥体等。

Ⅱ.几何体的变换

OpenSCAD可以对几何体进行平移、旋转、缩放等变换操作。

Ⅲ.几何体的组合

OpenSCAD可以将多个几何体组合成一个复合几何体，如并、交、差集等。

Ⅳ.几何体的参数化

OpenSCAD可以通过参数化操作来创建可重复使用的几何体，如通过变量来控制几何体的大小、位置、形状等。

Ⅴ.几何体的导入和导出

OpenSCAD可以导入和导出多种文件格式，如STL、OBJ、DXF等。

Ⅵ.脚本编程

OpenSCAD可以通过编写脚本来创建复杂的几何体形状，可以使用类似于C语言的语法来编写脚本。

1.4 本文主要工作

Ⅰ.将做工所需的目标切割体数字化

Ⅱ.利用这些数据设计和生成基于OpenGL的二维测试模型和基于OpenSCAD的三维测试模型

Ⅲ.开发碳素材料套料加工的交互式布局系统去实现对测试模型的操纵

Ⅳ.通过可制造性的验证算法来获得合理的布局结果

1.5 本文组织结构

本文共分为五章。

第 1 章 为绪论部分，介绍了昂贵碳素材料套料加工的研究背景以及国内外的研究现状。并对交互式系统涉及到的实现手段OpenGL和OpenSCAD进行了介绍。最后指明了本文所做的主要工作和本文的组织结构。

第 2 章 介绍了如何将现实物体坐标化，为第3章设计与生成测试模型做准备。

第 3 章 为碳素材料套料加工的交互式布局系统的实现过程。

第 4 章 详细介绍了可制造性的验证算法，阐述了算法的基本思想、算法所用的数据结构、算法实现的基本流程以及算法具体设计与实现。

第 5 章 为结论部分，对面向昂贵碳素材料套料加工的交互式布局系统的设计和实现工作进行总结。

第 2 章 目标切割体的数字化

将现实物体坐标化是设计和生成测试模型的基础，提供准确的数据，支持多种建模方法，保证模型的准确性和精度，建模者可以直接使用坐标数据来构建模型，而不需要进行繁琐的测量和计算。

2.1 原始碳素材料的数字化

原始碳素材料，即制作目标切割体的原始材料，碳素材料加工企业的某项目所在工厂提供的现场照片如图(2.1.1)所示。

图 2.1.1 原始碳素材料

从图中得知，原始碳素材料可近似为一种实心圆柱体。由于 OpenSCAD 可以使用基本的二维模型来创建复杂的三维模型，而本文设计的交互式系统中的三维模型也可以通过二维图形经过一系列变换得到，所以我们对其进行二维的坐标描述，即将实心圆柱体简化为矩形。

对原始碳素材料的坐标描述：

该原始碳素材料有 4 个顶点 S1(a1,b1)，S2(a2,b2)，S3(a3,b3)，S4(a4,b4)，如图(2.1.2)所示，且存在以下约束：

$$\begin{cases} S1S4 \perp S3S4; \\ S3S4 \perp S2S3; \\ S2S3 \perp S1S2; \\ S1S2 \perp S1S4; \\ S1S2 = S3S4; \\ S1S4 = S2S3 \end{cases}$$

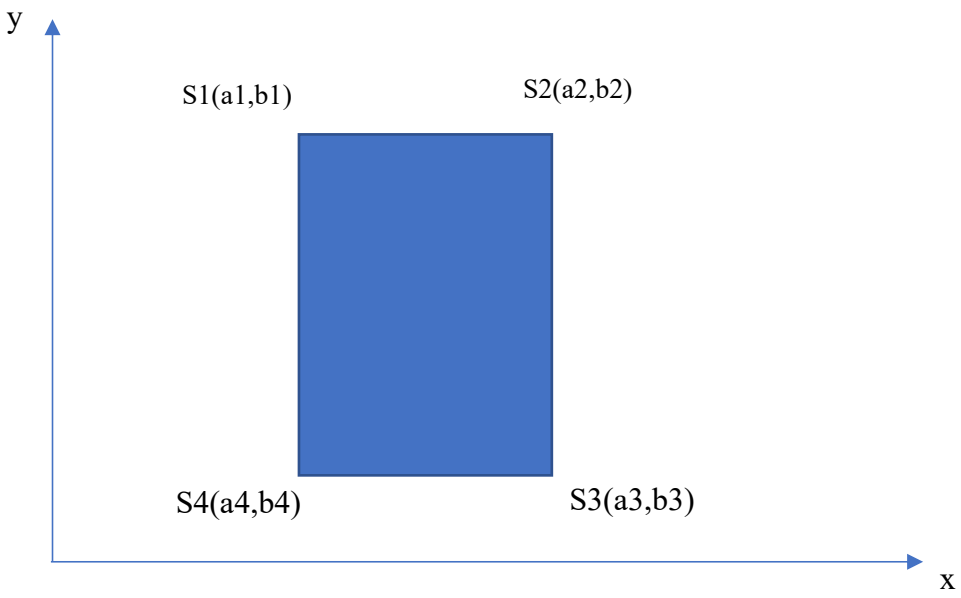


图 2.1.2 原始碳素材料的坐标描述

2.2 目标切割体的数字化

目标切割体，即从原始碳素材料切割出的最终产物。碳素材料加工企业的某项目所在工厂提供的现场照片如图(2.2.1)所示。

图 2.2.1 目标切割体

从图中得知，目标切割体可近似为一种空心旋转体薄片，由于是旋转体，所以是关于旋转轴对称的。由于 OpenSCAD 可以使用基本的二维模型来创建复杂的三维模型，而本文设计的交互式系统中的三维模型也可以通过二维图形经过一系列变换得到，所以我们对其进行二维的坐标描述，将空心旋转体薄片简化为剖面

图的左半边或右半边，如碳素材料加工企业的某项目所在工厂提供的剖面图(2.2.2)的左半边和右半边。

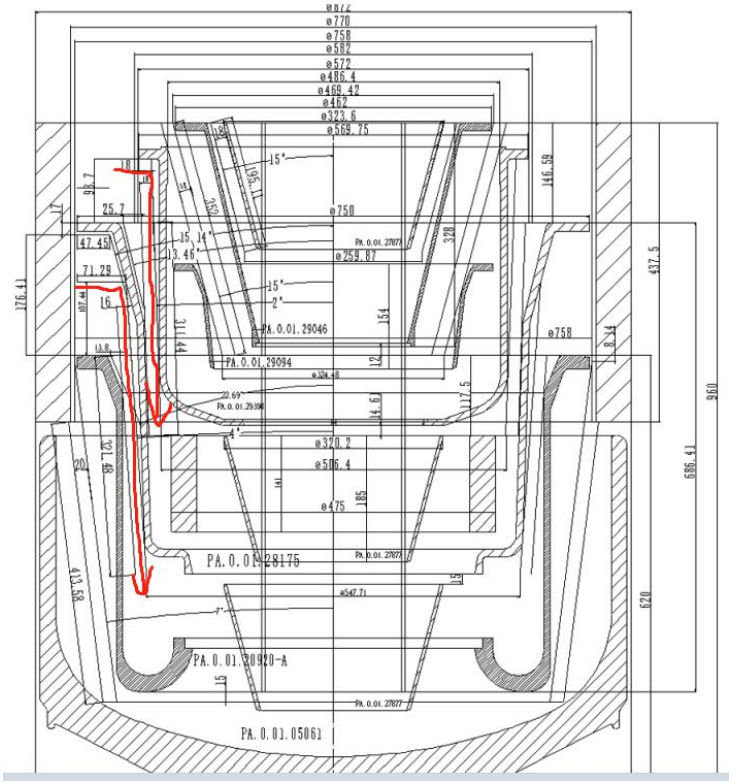


图 2.2.2 目标切割体剖面图

对目标切割体的坐标描述：

设目标切割体有 4 个顶点 T1(x1,y1)，T2(x2,y2)，T3(x3,y3)，T4(x4,y4)，如图(2.2.3)所示。

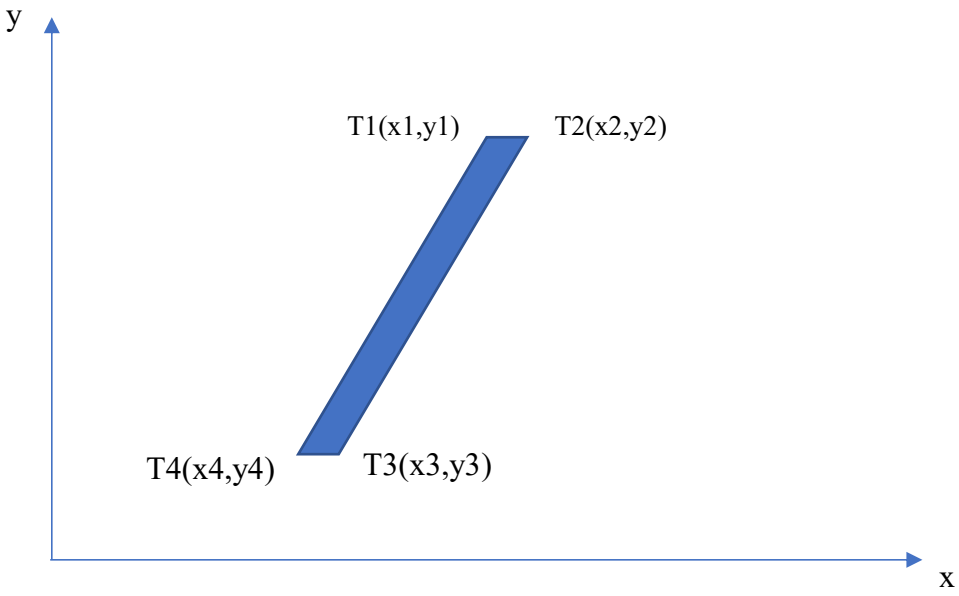


图 2.2.3 原始碳素材料的坐标描述

碳素材料加工企业的某项目所在工厂提供的剖面图实例(2.2.2)经过一系列化简得到目标切割体的平面图实例(2.2.4)，本文将以这 8 组目标切割体作为样例设计交互式系统。

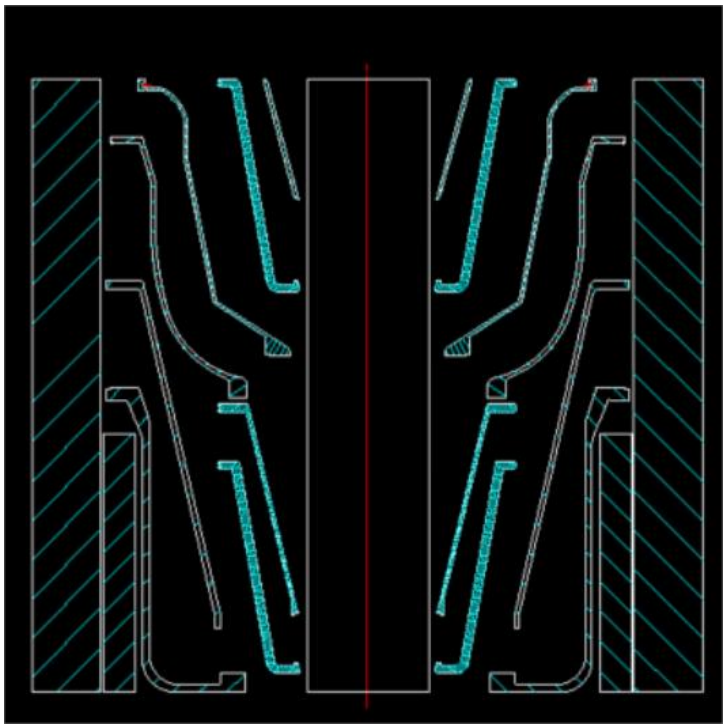


图 2.2.4 目标切割体的平面图

由于该平面图并没有准确的坐标描述，所以本文在方格背景下仿照作图(2.2.5)，以红色实线为 y 轴，以图片底部为 x 轴，一方格为一个单位长度。由于以 y 轴对称，所以只绘制了右半边，获得 8 组坐标描述如下表(2.2.1)，分别命名为 A,B,C,D,E,F,G 和 H，其中 C 和 D 的曲线部分将拟合成折线（粗加工不处理有弧度的切割体）。

表 2.2.1 8 组目标切割体的坐标点集合

组	坐标点集合
Apoints	{[3,24],[3.1,24],[4,28],[4.1,28]}
Bpoints	{[3,20],[4,20],[5,28],[6,28], [6,28.1],[4.9,28.1],[3.9,20.1],[3,20.1]]}
Cpoints	{[4,17],[5,17],[5,17.67],[7,19],[8.5,25], [8.5,27],[11,28],[11,28.1],[8.4,28.1],[8.4,25], [6.9,19.1],[3.8,17]}
Dpoints	{[6,14],[10,14],[10,23],[11,26],[12,26],[12,26.1],[11,26.1], [10,23.1],[6,14.1]}
Epoints	{[3,3],[3.1,3],[5,12],[6,12],[6,12.1],[4.9,12.1]]}
Fpoints	{[3,1],[4,1],[5.5,10],[6,10],[6,10.1],[5.4,10.1],[3.9,1.1], [3,1.1]}
Gpoints	{[7,3],[7.1,3],[7.1,4],[11,19],[12,19],[12,19.1],[10.9,19.1],[7,4]}
Hpoints	{[5,0],[10,0],[10,10],[11,13],[12,13],[12,13.1],[10.9,13.1],[9.9,10], [9.9,0.1],[6,0.1],[6,0.5],[5,0.5]}

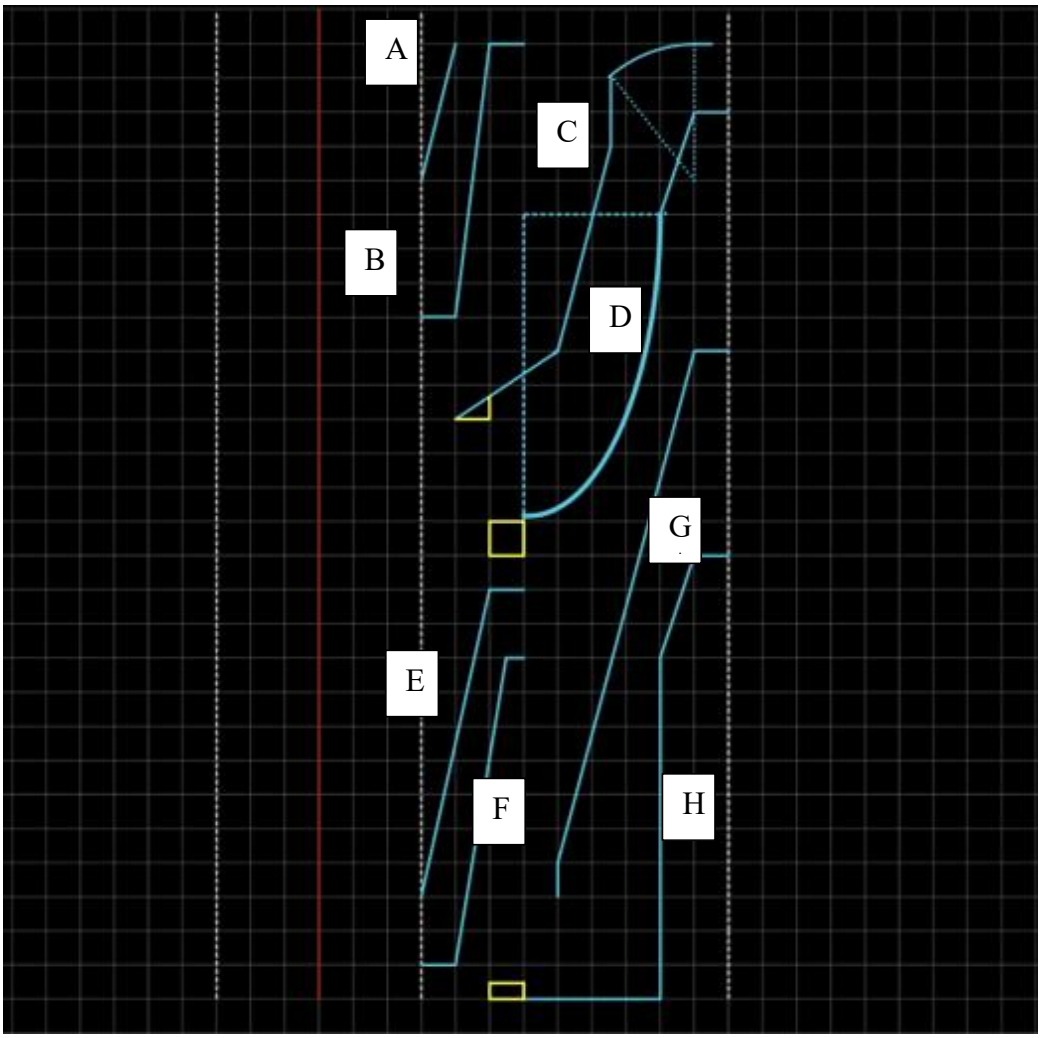


图 2.2.5 8 组目标切割体的方格图

第 3 章 碳素材料套料加工的交互式布局系统

3.1 编程语言、开发工具和配置环境

编程语言：C++

开发工具：Visual Studio

配置 OpenGL 环境：

首先安装 OpenGL 库文件，可以从官方网站或第三方网站下载，下载 glad 以及 glfw，根据系统下载 32 位。

其次安装支持 OpenGL 的开发环境，例如 Visual Studio、Code::Blocks 等。

然后在开发环境中配置编译器，将 OpenGL 库文件添加到编译器的链接库中。右键项目名称打开属性，点击 VC++目录，点击包含目录，打开之后选择编辑，勾选从父级项目默认设置继，选择刚刚下载的 glad 包和 glfw 包中的 include 和 lib 文件夹。点击链接器附加依赖项输入 opengl32.lib 以及 glfw.lib。之后打开 glad 文件夹里的 src 文件，将源码复制到项目的源文件中。

再编写 OpenGL 程序，包括初始化 OpenGL 环境、创建窗口、设置视角、绘制图形等。

最后编译和运行程序，将 OpenGL 程序编译成可执行文件，运行程序并查看效果。

3.2 模型绘制

同时具有二维和三维图像的交互式系统有以下优点（表 3.2.1）。

表 3.2.1 具有二维和三维图像的交互式系统的优点

优点	内容
更直观的展示方式	二维图像可以展示平面上的信息，而三维图像可以展示物体的立体结构，结合起来更直观，使得用户更容易理解和掌握
更全面的信息呈现	二维图像可以展示物体的表面信息，而三维图像可以展示物体的内部结构，结合起来使得用户更全面地了解物体的特征和属性
更灵活的交互方式	三维图像可以跟随二维图像一起运动，二维图像可以通过鼠标、键盘等方式进行交互

3.2.1 基于 OpenGL 的二维测试模型的设计和生成

定义结构体 MyPoint，存储一个点的横坐标 x 和纵坐标 y。

```
struct MyPoint
{
double x;
double y;
};
```

定义以 MyPoint 为数据类型的数组 pt1,pt2,pt3,pt4,pt5,pt6,pt7 和 pt8，存储 8 组目标切割体的坐标点集合，以 pt1 为例：

```
vector<MyPoint> pt1 ={};
```

使用 for 循环遍历 pt1 数组，通过 glVertex2f()函数绘制目标切割体的右半边和左半边，以 pt1 为例：

```
glBegin(GL_POLYGON);
 for (int i = 0; i < pt1.size(); i++)
 {
  glVertex2f(pt1[i].x, pt1[i].y);
 }
 glEnd();
 glBegin(GL_POLYGON);
 for (int i = 0; i < pt1.size(); i++)
 {
  glVertex2f(-pt1[i].x, pt1[i].y);
}
glEnd();
```

将 8 个模块的代码全部完成后进行 display()，可得出如下图(3.2.1)的总模型，方便在交互式系统中加载。

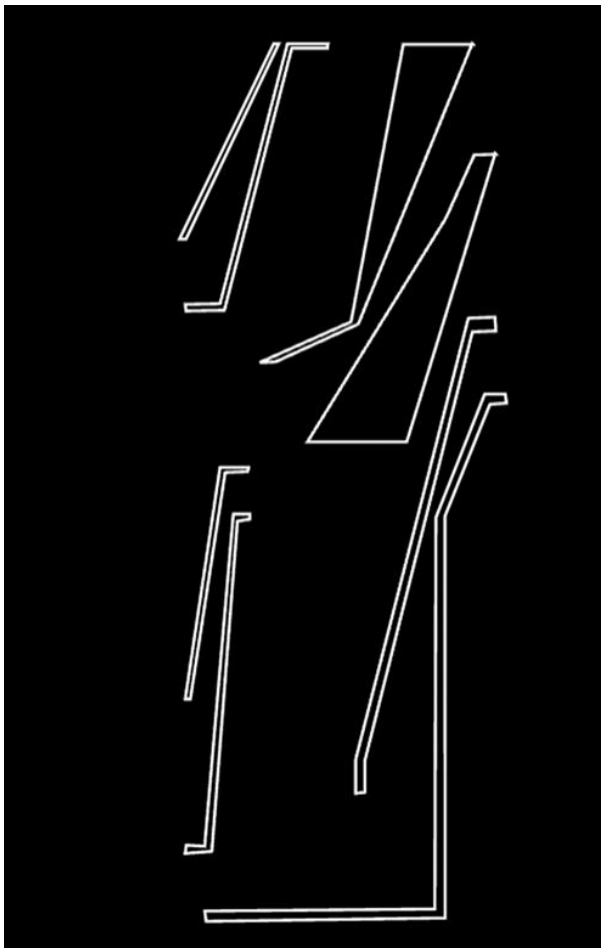


图 3.2.1 8 组目标切割体的 2D 模型

3.2.2 基于 OpenSCAD 的三维测试模型的设计和生成

在 OpenSCAD 中，module 是一种定义可重用 3D 模型的方式。可以将一组 3D 对象封装在一个 module 中，然后在需要使用这些对象的地方调用该 module，从而避免重复编写相同的代码。

由于有 8 组目标切割体的坐标点集合，所以使用 module 方式定义 8 个模块 one,two,three,four,five,six,seven 和 eight，以 one 为例：

```
1. module one(){…}
2. one();
```

在 OpenSCAD 中，translate 函数用于将对象沿着指定的方向移动一定的距离。由于目标切割体是轴对称体，故不需要移动一定距离，所以在这 8 个模块中均为 translate([0,0,0])。

在 OpenSCAD 中，rotate_extrude 命令用于将一个 2D 图形沿着旋转轴旋转并拉伸成为一个 3D 模型。rotate_extrude 的语法如下：

```
rotate_extrude(angle = 360, convexity = 10) {
  // 2D 图形定义
}
```

其中，angle 参数指定旋转角度，默认为 360 度，即完整旋转一周。convexity 参数指定模型的凸度，即模型表面的平滑程度，默认为 10。

在 OpenSCAD 中，polygon()命令用于创建一个多边形。该命令可以用于创建各种形状的多边形，如正多边形、不规则多边形等。polygon()的语法如下：

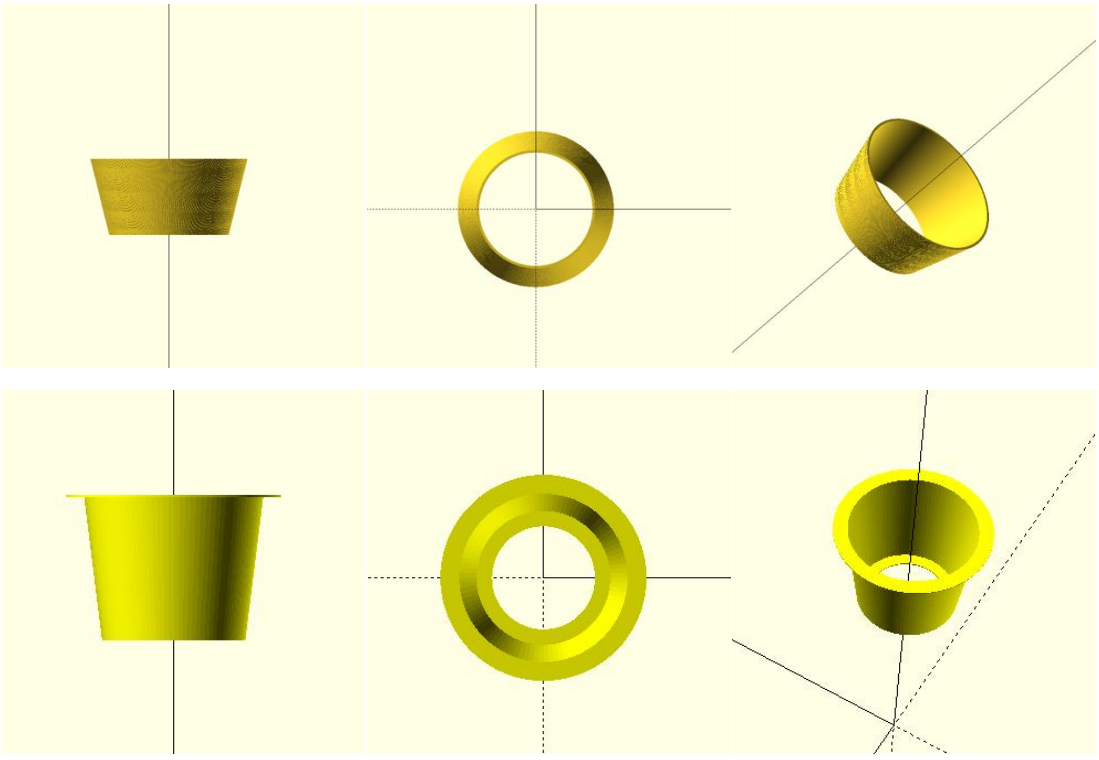
```
polygon(points, paths);
```

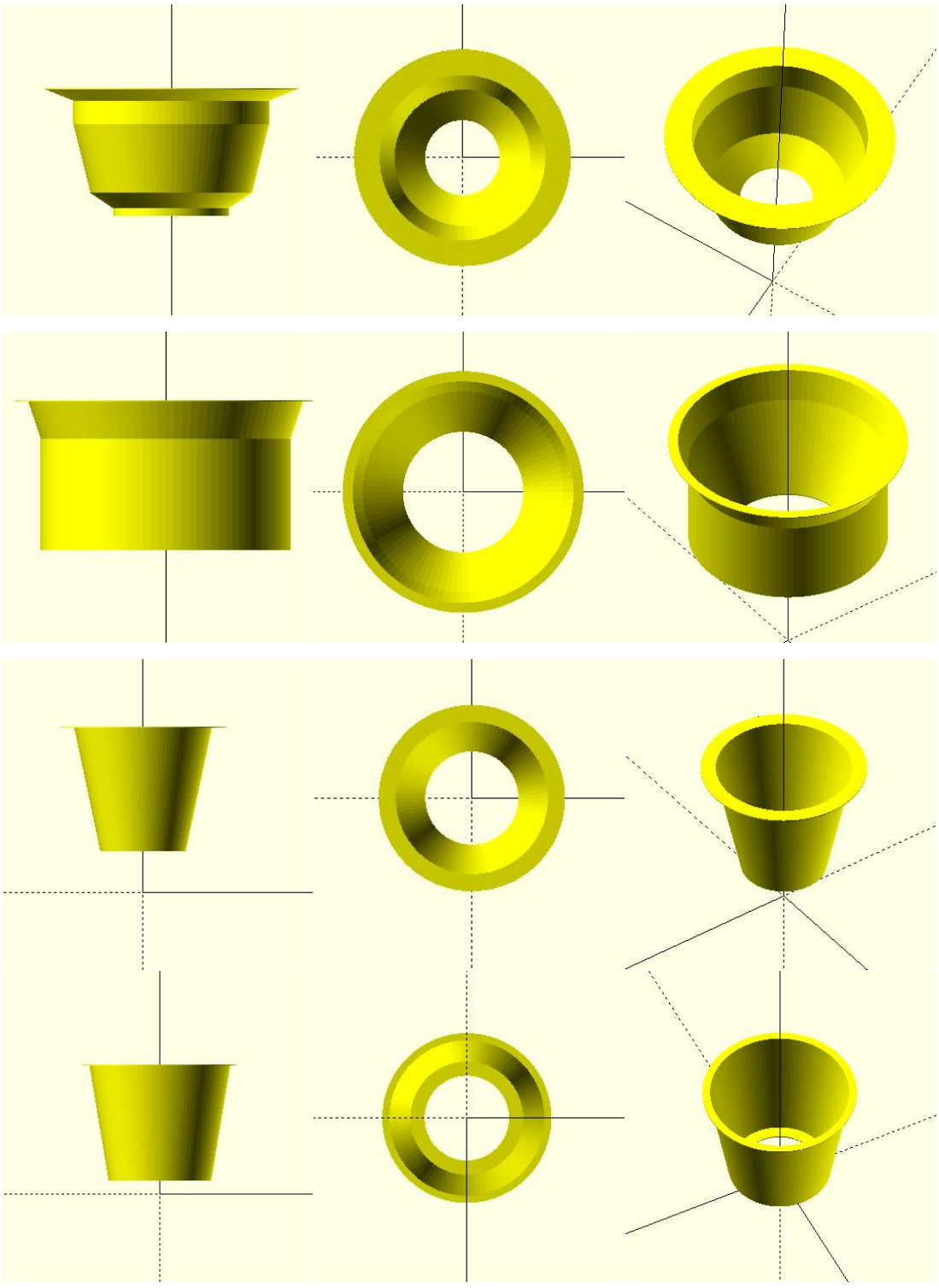
其中，points 参数是一个包含多边形各个顶点坐标的数组，每个顶点坐标用一个二元组表示。paths 参数是一个包含多边形各个边的顶点索引的数组，每个边用一个整数数组表示。

以 one 为例：

```
rotate_extrude($fn=100)
        polygon(
        points=[…]
        );
```

将 8 个模块的代码全部完成后进行 render，可得出如下图(3.2.2)的总模型，并导出.stl 文件[15]，方便在交互式系统中加载。





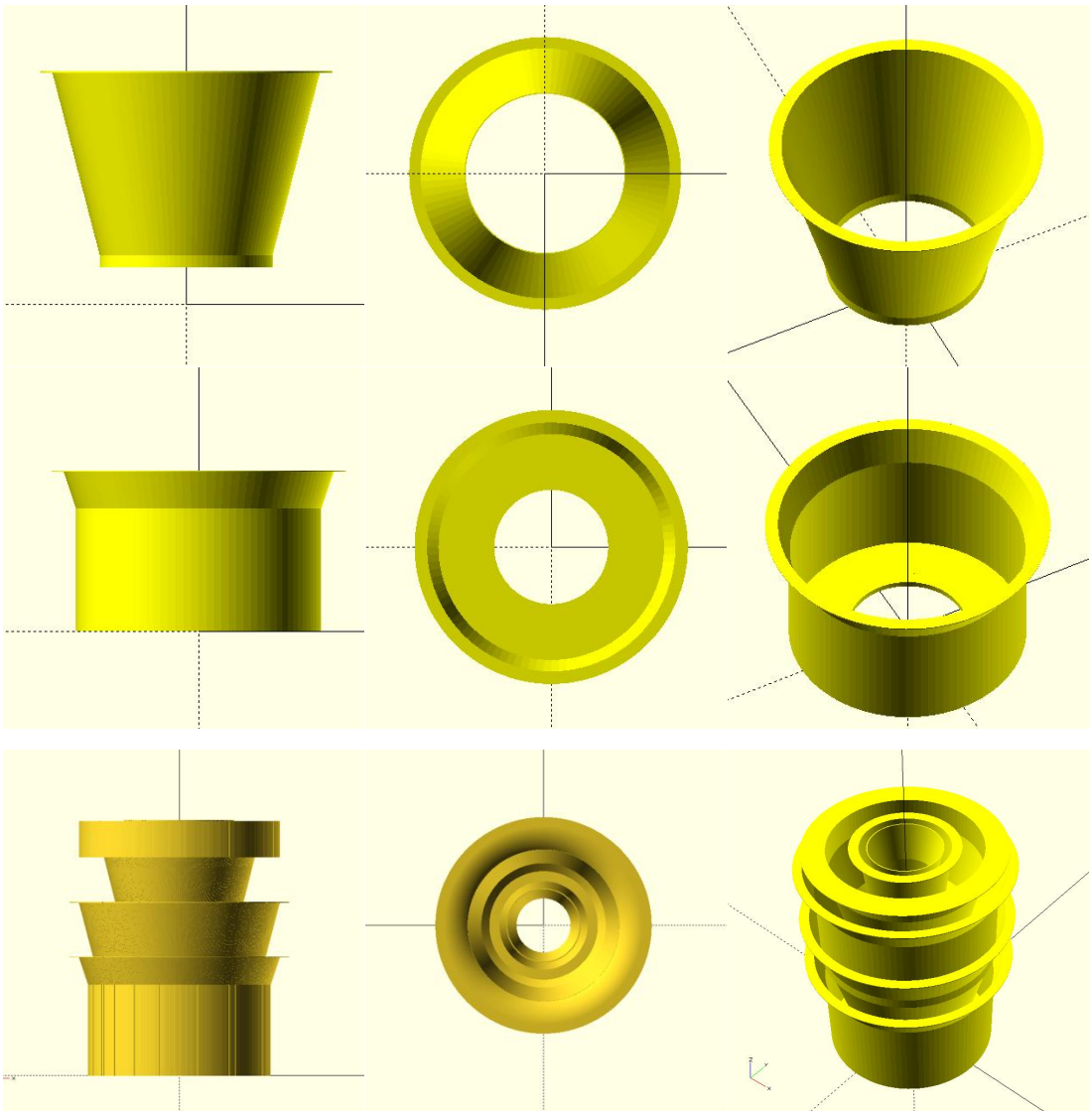


图 3.2.2 8 组目标切割体的独立 3D 模型和总体模型

3.3 操作和系统功能(表 3.3.1)

在本小节中会列举一些主要算法，例如水平/垂直交叉点数判别法等，用于判断判断点是否在多边形内部。

表 3.3.1 操作和系统功能及其详情

操作和功能	所需
绘制 2d 图形	OpenGL 相关函数等
鼠标响应事件	OpenGL 相关命令等
拖拽 2d 图形	鼠标响应事件+判断点是否在多边形内部算法等
键盘响应事件	OpenGL 相关命令等
缩放 3d 物体	OpenGL 相关函数等
判断 2d 图形之间是否相交，如果相交则判定无法切割且刀具无法布局，否则判定可切割且输出刀具布局	可制造性的验证算法(需要判断点是否在多边形内部算法)等
2d 图形和 3d 物体的实时移动	读取 3d 物体的.obj 文件并更新

3.3.1 绘制 2d 图形(同 3.2.1)

3.3.2 鼠标响应事件[16]

在 OpenGL 中，鼠标响应事件可以通过 GLUT 库来实现。GLUT 库提供了一些回调函数，可以在特定的事件发生时被调用，表(3.3.2)是一些常用的鼠标响应事件和对应的回调函数。

表 3.3.2 一些常用的鼠标响应事件和对应的回调函数

鼠标响应事件	触发条件	对应的回调函数
鼠标点击事件	当用户点击鼠标按钮时	Void glutMouseFunc(void (*func)(int button, int state, int x, int y)) 。其中，button 参数表示按下的鼠标按钮，state 参数表示按钮的状态（按下或释放），x 和 y 参数表示鼠标点击的位置。
鼠标移动事件	当用户移动鼠标时	void glutMotionFunc(void (*func)(int x, int y));其中，x 和 y 参数表示鼠标移动的位置。
鼠标滚轮事件	当用户滚动鼠标滚轮时	void glutMouseWheelFunc(void (*func)(int button, int dir, int x, int y));其中，button 参数表示滚轮所在的鼠标按钮，dir 参数表示滚轮的方向（正向或反向），x 和 y 参数表示鼠标滚轮的位置。

通过注册这些回调函数，可以实现鼠标响应事件的处理。在回调函数中，可以根据事件的类型和参数来实现相应的操作。以下是本文设计的交互式系统的有关鼠标操作的主要函数，实现的功能如表(3.3.3)。

```
1. void mymouse(int button, int state, int x, int y)
2. {…
3.  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
4.  {…
5.  }
6. if (state == GLUT_UP && button == GLUT_WHEEL_UP)//滚轮往上滚
7. {…
8. }
9.  else if (state == GLUT_UP && button == GLUT_WHEEL_DOWN)//滚轮往下滚动
10.      {…
11.      }
12.
13.       else if (state == GLUT_UP)
14.       {…
15.       }
16.
17.      }
```

表 3.3.3 鼠标操作和对应的功能

鼠标操作	对应的功能
滚轮向上滚动	放大三维模型
滚轮向下滚动	缩小三维模型
左键按下且选中二维图像并拖动	移动二维图像

3.3.3 拖拽 2d 图形

判断点是否在多边形内部的算法 PtInPolygon()，具体算法解析见第四章。

3.3.4 键盘响应事件[17]

在 OpenGL 中，键盘响应事件可以通过 GLUT 库来实现。GLUT 库提供了一些回调函数，可以在特定的事件发生时被调用。以下是一些常用的键盘响应事件和对应的回调函数。

表 3.3.4 一些常用的键盘响应事件和对应的回调函数

键盘响应事件	触发条件	对应的回调函数
普通键盘事件	当用户按下或释放普通键盘键时	void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));其中，key 参数表示按下或释放的键盘键的 ASCII 码值，x 和 y 参数表示键盘事件发生的位置。
特殊键盘事件	当用户按下或释放特殊键盘键时	void glutSpecialFunc(void (*func)(int key, int x, int y));其中，key 参数表示按下或释放的特殊键盘键的键值，x 和 y 参数表示键盘事件发生的位置。
持续键盘事件	当用户按住键盘键时	void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));

在回调函数中，可以根据事件的类型和参数来实现相应的操作。以下是本文设计的交互式系统的有关键盘操作的主要函数，实现的功能如表(3.3.5)。

```
1.  void KeyboardEvent(unsigned char key, int x, int y)
2.  {
3.   switch (key)
4.   {
5.   case 'W'://上移动
6.    m_tranlate[1] += 0.1;
7.    break;
8.   case 'S'://下移动
9.    m_tranlate[1] -= 0.1;
10.   break;
11.  case 'A'://左移动
12.   m_tranlate[0] -= 0.1;
13.   break;
14.  case 'D'://右移动
15.   m_tranlate[0] += 0.1;
16.   break;
17.  case 'Z':
18.   m_model = m_model == 0 ? 1 : 0;
19.   break;
20.  }
```

表 3.3.5 键盘操作和对应的功能

键盘操作	对应的功能
按下“W”	三维模型向上移动
按下“A”	三维模型向左移动
按下“S”	三维模型向下移动
按下“D”	三维模型向右移动

3.3.5 缩放 3d 物体

glScalef()是 OpenGL 中的一个函数，用于对当前矩阵进行缩放变换。它的原型如下：void glScalef(GLfloat x, GLfloat y, GLfloat z);其中，x、y、z 分别表示在 x、y、z 三个方向上的缩放因子。如果缩放因子为正数，则物体会被放大；如果缩放因子为负数，则物体会被翻转。

glScalef()函数会将当前矩阵与一个缩放矩阵相乘，从而实现缩放变换。缩放矩阵的形式如下：

$$
\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

其中，x、y、z 分别为缩放因子。glScalef()函数会将当前矩阵与上述矩阵相乘，从而实现缩放变换。本文交互式系统中对三维模型的缩放为：

```
1. glScalef(m_scale, m_scale, m_scale)
```

其中 m_scale 是对三维模型的缩放比例。这会将当前矩阵与一个缩放矩阵相乘，从而将物体在 x，y，z 方向上缩放对应的比例（本文为等比例缩放，所以 x=y=z）。

3.3.6 判断二维图形之间是否相交(见第四章)

3.3.7 二维图形和三维模型的实时移动

只要将表达三维模型结构和数据的文件在 OpenGL 中加载并更新，即可使二维图形跟随三维模型实时移动。本文使用的是 ASCII 格式的 STL 文件类型来表达三维模型。

3.3.7.1 ASCII 格式的 STL 文件结构[18]

ASCII 格式的 STL 文件是一种文本文件，每个三角形面片都用一个 facet 关键字来表示，其格式如下：

```
solid filename  //文件路径及文件名
facet normal x y z//三角片面法向量的 3 个分量
vertex x1 y1 z1 x2 y2 z2 x3 y3 z3 //三角片面三个顶点的坐标
endfacet //三角面片完毕
endsolid filename //整个文件结束
```

其中，object_name 是物体的名称，x、y、z 是法向量的三个分量，x1、y1、z1 等是三角形的三个顶点坐标。

3.3.7.2 OpenGL 加载和绘制 STL 模型

在 OpenGL 中加载和绘制 STL 模型，需要进行以下步骤：

1.读取 STL 文件数据：使用 STL 文件解析库读取 STL 文件数据，并将其转换为 OpenGL 可用的数据结构。

定义一个类 STLModel,用来加载和显示模型：

```
class STLModel
{
public:
 //构造函数 1
 //构造函数 2
 //析构函数
  bool LoadStlFile(const char* stlFileName);//加载 STL 模型文件并解析
  void Draw(int model);//绘制模型

protected:
 struct RenderTri//一个三角形所有数据结构体
 {
  M3DVector3f m_Normal; //法线
  M3DVector3f m_Vertx1;//三角形第一个顶点
  M3DVector3f m_Vertx2;//三角形第二个顶点
  M3DVector3f m_Vertx3;//三角形第三个顶点
 };
private:
   std::vector<RenderTri> m_MeshTris;   //保存加载的三角形数据
   M3DVector3f m_MaxPos;//记录最大的 X Y Z 点 方便平移 缩放用来最佳显示
   M3DVector3f m_MinPos;//记录最小的 X Y Z 点 方便平移 缩放用来最佳显示
};
```

类 STLModel 中的函数 LoadStlFile()实现 STL 文件对应数据的加载。具体思路如图(3.3.1)，代码核心如下。

图 3.3.1 STL 文件对应数据加载的函数思路

```
1.  bool STLModel::LoadStlFile(const char* stlFileName)
2.  {
3.   int Triangle_num = 0;//记录读取的三角形个数
4.   while (fgets(buffer, 133, file))//开始读取每一个三角形数据
5.   {
6.    lineno++;//记录读取的行数 加 1
7.    RenderTri tri;//三角形结构体数据，包括 3 个顶点坐标和 1 个法线坐标
8.   //读取法线坐标
9.   // 过滤掉 outer loop
10. //读取第一个顶点坐标
11. //记录最大值和最小值，方便平移和缩放 让模型在视图正中间显示
12.  {
13.   //记录坐标点
14. …
15.   for (int i = 0; i < 3; i++)
16.   {
17.   //比较并记录最大值
18.   //比较并记录最小值
19.   }
20.  }
21.  m_MeshTris.push_back(tri);//把三角形数据保存到容器中
22. return true;
23. }
```

2.创建 OpenGL 对象：使用 OpenGL 函数创建顶点缓冲区、索引缓冲区、纹理缓冲区等 OpenGL 对象。

3.填充 OpenGL 对象[19]：将 STL 文件中的顶点、法向量、纹理坐标等数据填充到 OpenGL 对象中。

类 STLModel 中的函数 Draw()实现对应数据的填充。

这个函数的主要思路是绘制 STL 模型，具体思路如图(3.3.2)，代码核心如下。

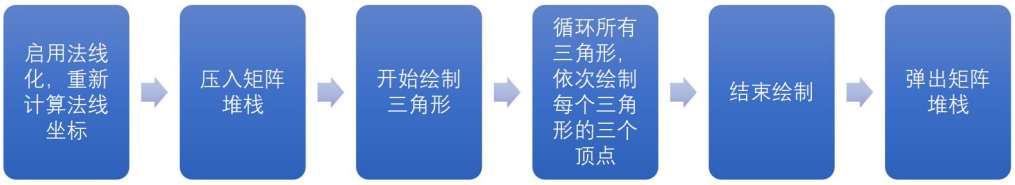


图 3.3.2 对应数据填充的函数思路

```
1.  void STLModel::Draw(int model)//绘制模型
2.  {
3.   glEnable(GL_NORMALIZE);//重新计算法线坐标
4.   glPushMatrix();//压栈
5.   …
6.   glBegin(GL_TRIANGLES);//绘制三角形
7.   if (model == 0)
8.   {
9.    for (size_t i = 0; i < m_MeshTris.size(); i++)//循环所有三角形
10.  {
11.   //第一个点数据
12.  //第二个点数据
13.  //第三个点数据
14.  }
15. }
16. glEnd();//结束绘制
17. glPopMatrix();//出栈
18.}
```

4.渲染 OpenGL 对象：使用 OpenGL 函数将 OpenGL 对象渲染到屏幕上。

3.4 数据输入输出(表 3.4.1)和刀具布局

表 3.4.1 数据输入和输出

输入	输入方式	输出	输出方式
二维图像	代码内的结构体数组	是否碰撞，若无碰撞给出刀具布局方案	命令行
三维模型	导出.stl 文件并在交互式系统中读取	是否碰撞，若无碰撞给出刀具布局方案	命令行

在判断目标切割体并未发生碰撞之后，进行刀具布局方案的判断，图(3.4.1)是刀具的 4 个种类，所有刀具不计厚度等因素，仅代表所处位置。

本文目前完成了空心刀、直刀和底刀的判断和布局，斜刀在二维平面内由于本身占据的空间与角度等不定因素有关，难以判断和布局，所以暂时搁置。

由于斜刀的不存在，故只能用多段直刀代替目标切割体中间折线段的切割，即贴合二维图形边缘即为直刀的布局。

1.空心刀用于掏空圆柱体，其母线是平行于 y 轴的，横坐标即为空心刀的半径。空心刀的半径改变取决于切割次序，将每组目标切割体的横坐标最小的顶点提取出来并对其横坐标从小到大依次排序，排序算法采取冒泡排序算法[20]，顺序为空心刀半径的改变过程，顺序对应的横坐标值为空心刀半径。

2.底刀用于削平目标切割体的顶端和底端，判断方法为相邻两个坐标(纵坐标最高或次高或最低或次低)的纵坐标是否相等，即筛选掉目标切割体的中部水平段。如果满足条件，则摆放位置为对应纵坐标的平行于 x 轴的直线，深度为横坐标的差值。

3.直刀目前用于切割目标切割体的所有内部边，布局直接贴合内部边。

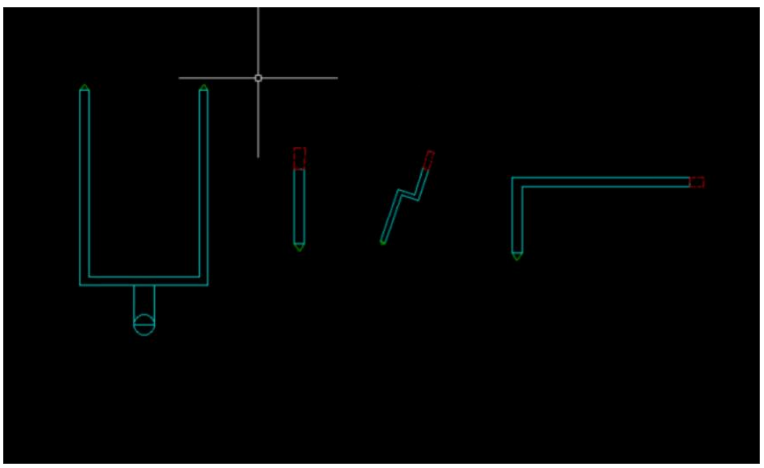


图 3.4.1 4 种刀具(从左往右依次命名为“空心刀”、“直刀”、“斜刀”和“底刀”)

部分布局情况的判断如图(3.4.2-3.4.6)。

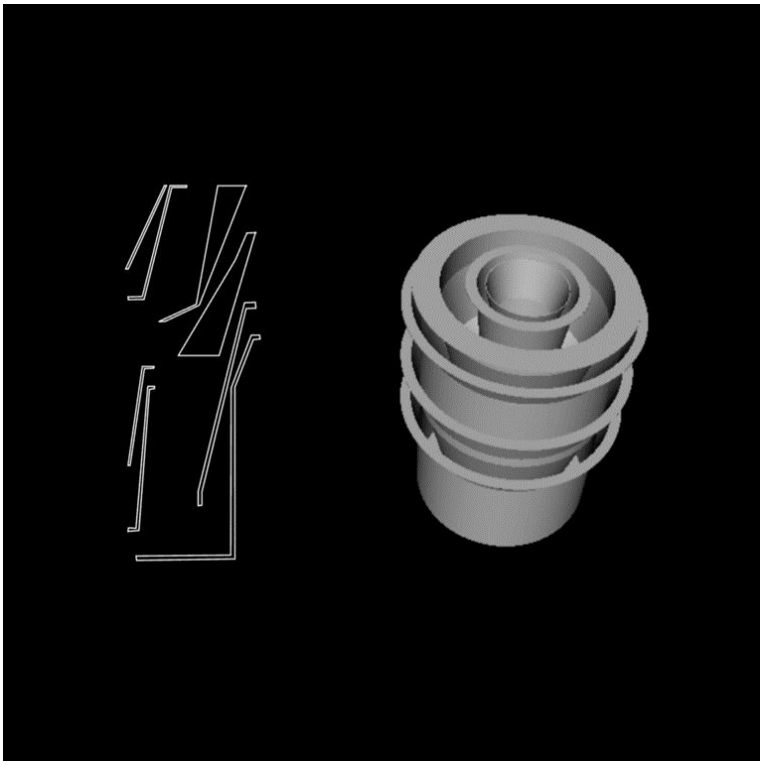


图 3.4.2 布局 1 的二维图像和三维模型的交互界面

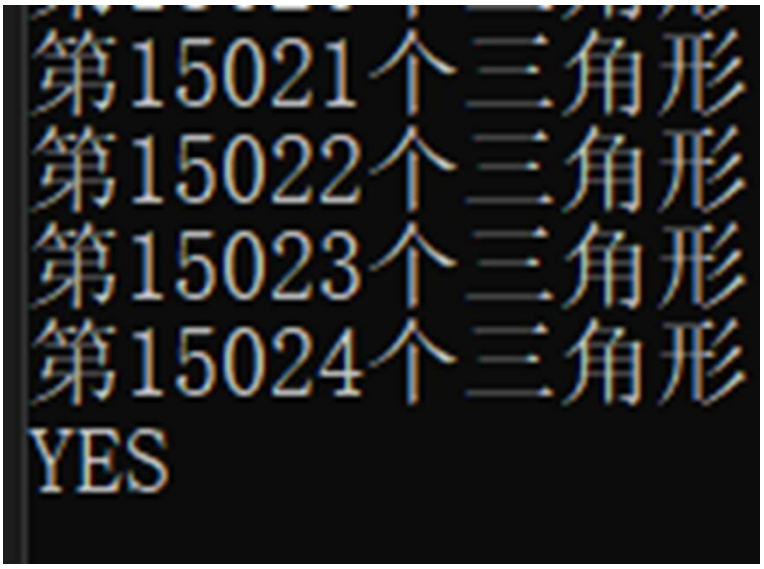


图 3.4.3 布局 1 是否成立的判断

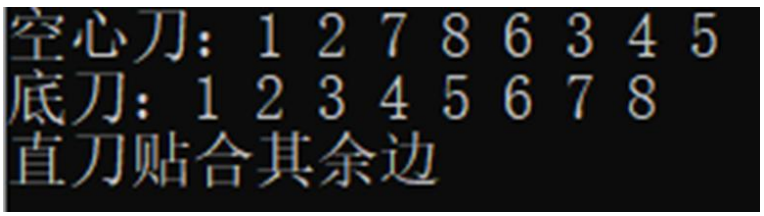


图 3.4.4 布局 1 的输出

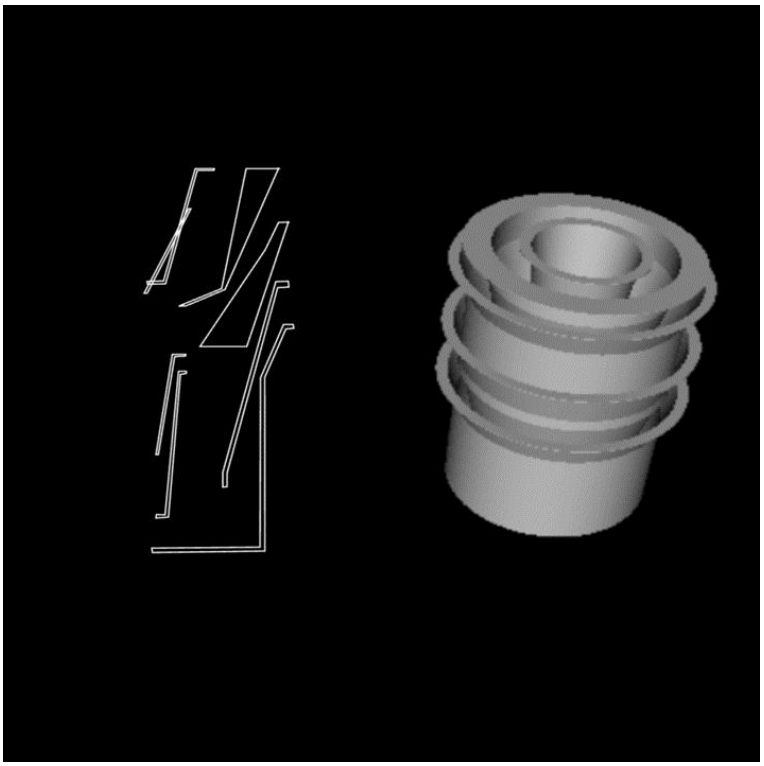


图 3.4.5 布局 2 的二维图像和三维模型的交互界面

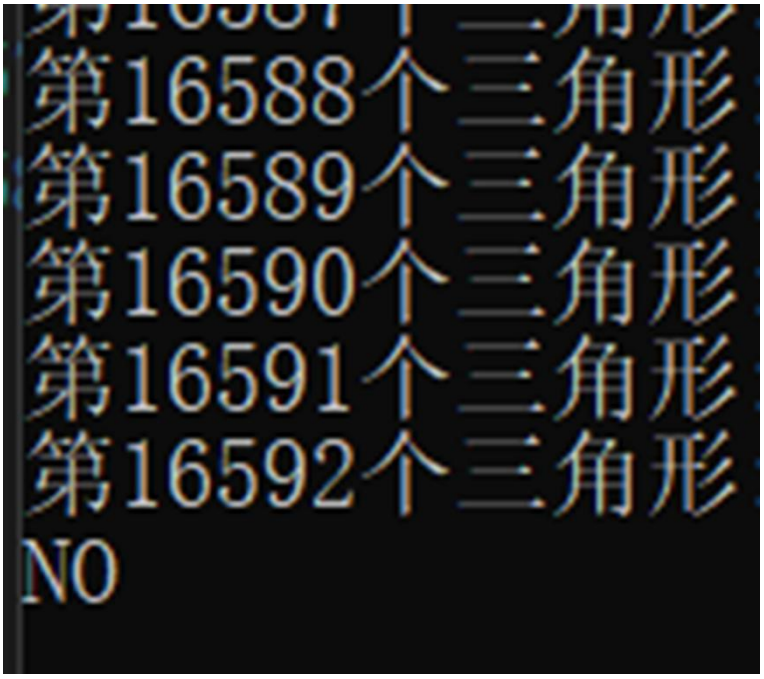


图 3.4.6 布局 2 是否成立的判断

第 4 章 可制造性验证算法

关于目标切割体是否可以被成功切割的问题，可以简化为判断两个多边形是否相交。而两个多边形是否相交取决于一个多边形的所有线段是否相交于另一个多边形，又因为线段是点组成的，所以只要知道在同一平面内，如何判断点是否在多边形上，再遍历线上所有的点即可，这里提供了一种算法称作水平/垂直交叉点数判别法[21]，即射线法。

4.1 算法基本思想

如图(4.1.1)所示，设红色实心点为点 P，已知点 P 在多边形内部。过点 P 向左做一条水平射线，与多边形的部分边存在粉色实心点 5 个；过点 P 向右做一条水平射线，与多边形的部分边存在粉色实心点 3 个。

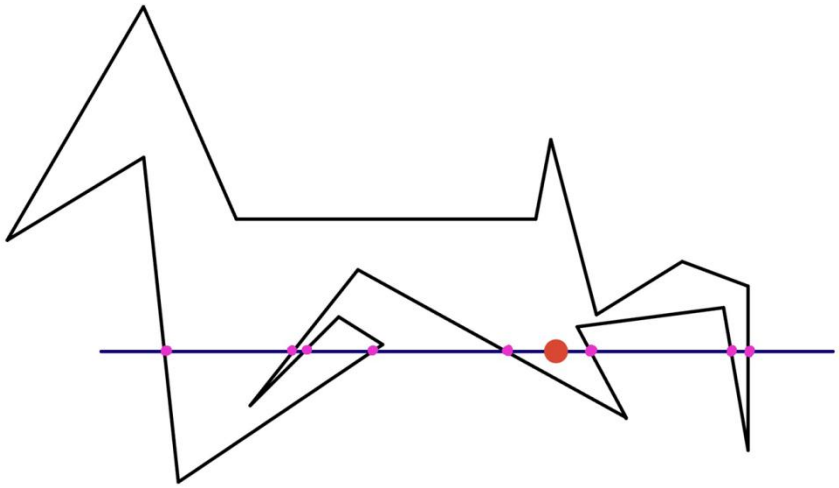


图 4.1.1 算法的解释样例

广泛地，若 P 在多边形内部，则过 P 的水平射线与多边形的交点个数必为奇数；若 P 在多边形外部，则过 P 的水平射线与多边形的交点个数必为偶数。

所以，我们可以按照顺序遍历多边形的每条边，求出交点的总个数。有边(P1,P2)，其中有 3 种特殊情况见表(4.1.1)。

表 4.1.1 3 种特殊情况及其处理方法

情况	处理方法
射线正好穿过 P1/P2,则 2 个交点	如果 P 的从坐标与 P1,P2 中较小的纵坐标相同，忽略
射线水平，则无交点或无数个交点	忽略
射线竖直，P 的横坐标小于 P1,P2 的横坐标，则必然相交	无

当可以判断一个点是否在多边形内部时，便可以通过“点动成线”的规则，遍历一条线段上所有的点，分别判断所有点是否在多边形内部。如果存在一个点在多边形内部，则这条线段与该多边形相交；如果任意一个点都不在多边形内部，则这条线段不与该多边形相交。

当可以判断一条线段是否与多边形相交时，便可以通过“线动成面”的规则，遍历一个多边形上所有的边，分别判断所有线段是否与另一个多边形相交，即可以判断目标切割体是否会发生碰撞。如果存在一条线段与另一个多边形相交，则多边形之间是相交的，目标切割体是相交的，是发生碰撞的，是无法制造出来的，刀具更是无法布局；如果任意线段都不与另一个多边形相交，则多边形之间是不相交的，目标切割体是不相交的，是没有发生碰撞的，是可以制造出来的，刀具是可以布局的。

4.2 算法具体设计与实现

开始输入点和多边形，以已知点为起点，无穷远处为终点做出 x 轴的平行线 a，通过 for 循环遍历输入过的多边形的每一条边 b，判断该边是否与 x 轴平行。如果平行，则继续 for 循环；如果不平行则可以判断已知点是否在此时遍历到的多边形的边上。如果 a 与 b 相交，则相交点的个数加一；如果不相交则继续 for 循环。最后用求余运算符%判断相交点个数是奇数还是偶数，其中 0 也算在偶数之内。如果相交点的个数是奇数，则该点在多边形内部；如果相交点的个数是偶数，则该点在多边形的外部。判断完毕。

第 5 章 结束语

5.1 本文总结

本文实现了简单的面向昂贵碳素材料套料加工的交互式布局系统，其中实现了一个目标切割体的碰撞检测器，用基于水平/垂直交叉点数判别算法的程序判断碰撞，用基于 OpenGL 的方法加载了二维坐标，贴合二维图形生成刀具，并实现了键鼠等基本交互，还深度解析了 ASCII 的 STL 文件格式，在 OpenGL 中提供了直观的 openSCAD 建立的三维模型，满足用户的基本需求，提升用户的做工效率。

5.2 工作展望

首先将手动化交互式布局系统升级为自动化交互式布局系统。自动化的布局方案相比手动布局有如表(5.2.1)所示优点。

表 5.2.1 自动化的布局方案的优点

优点	内容
提高效率	快速生成布局，减少手动调整布局的时间和精力
减少错误	手动布局容易出现错误，而自动化布局方案可以减少这些错误，提高布局的准确性
更加灵活	自动化布局方案可以根据不同的需求进行调整，而手动布局则需要重新计算和调整
可维护性	自动化布局方案可以更容易地进行维护和修改，而手动布局则需要重新计算和调整

总之，自动化的布局方案可以提高开发效率、减少错误、提高灵活性和可维护性，是不可或缺的一部分。

其次完成第 3 种刀具“斜刀”的刀具布局方案，思路是将斜刀看作是一种绕指定点旋转的多边形，并进行碰撞检测。

最后用 NX/UG 等做出数控的三维动画展示，使初学者用户更加直观地认识该交互式系统。

参考文献

[1]A. K. Geim， K. S. Novoselov(2007). Graphene: Status and Prospects. Science.

[2]李宏伟, 陈晓东. 一种基于贪心算法的套料算法[J]. 计算机工程与应用, 2009, 45(9): 1-3.

[3] 姚宏, 王建华, 马晓宇. 基于遗传算法的零件套料优化研究[J]. 机械设计与制造, 2017, 12(1): 1-5.

[4] 张磊, 王建华, 马晓宇. 基于改进遗传算法的零件套料优化研究[J]. 机械设计与制造, 2018, 12(1): 1-6.

[5] 刘卫东. 基于虚拟现实技术的数控加工交互式系统设计与实现[D]. 河南：河南理工大学, 2017.

[6] Liu, Y., & Li, K. (2019). A Review of Human-Computer Interaction Techniques for Virtual Reality. Journal of Physics: Conference Series, 1184(1), 012032.

[7] Wang, Y., Zhang, Y., & Li, J. (2018). Research on the Optimization of CNC Machining Parameters Based on the Taguchi Method. Journal of Physics: Conference Series, 1065(1), 012064.

[8] Wang, Y., Li, J., & Zhang, Y. (2019). Automatic Layout Scheme of Parts Based on Geometric Algorithms. Journal of Mechanical Engineering, 55(2),

[9] Chen, Y., & Li, Y. (2018). A greedy algorithm for two-dimensional rectangular packing problem. Journal of Industrial and Management Optimization, 14(4), 1391-1404.

[10]J. L. González-Velarde “A genetic algorithm for the two-dimensional rectangular packing problem”， Computers & Operations Research。

[11] M. M. Rahman, M. A. Hossain, and M. A. H. Mithu“Simulated Annealing Algorithm for Multi-Objective Optimization in CNC Turning of AISI 1045 Steel”.

[12] A Tabu Search Algorithm for the Vehicle Routing Problem with Time Windows”，作者：M. Gendreau, G. Laporte, F. Semet, R. Séguin，发表时间：1996 年，期刊名称：Transportation Science。

[13]“OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5”，作者：John Kessenich, Graham Sellers, Dave Shreiner，发表时间：2016 年，出版社：Addison-Wesley Professional。

[14]“OpenSCAD: The Programmers Solid 3D CAD Modeller”，作者：Marius Kintel, Clifford Wolf，发表时间：2011 年，出版社：ACM。

[15]“STL File Format for 3D Printing”，作者：M. A. Al-Badri, M. A. Al-Saedi，发表时间：2017 年，出版社：International Journal of Computer Applications。

[16] “A Study on Mouse Response Time in Human-Computer Interaction” by Y. Kim and S. Lee, published in the International Journal of Human-Computer Interaction in 2013.

[17] “The Effects of Keyboard Response Time on User Performance” by J. Smith and K. Johnson, published in the Journal of Human-Computer Interaction in 2012.

[18] “A Comparative Study of Binary and ASCII STL File Formats for Rapid Prototyping” by S. K. Singh and A. K. Singh, published in the International Journal of Advanced Research in Computer Science and Software Engineering in 2014.

[19] “Real-time Rendering of Large-scale STL Models using OpenGL” by J. Zhang, Y. Liu, and Y. Zhang, published in the Journal of Computer-Aided Design & Computer Graphics in 2015.

[20] “An Improved Bubble Sort Algorithm for Sorting Large Data Sets” by S. S. Patil and S. S. Kulkarni, published in the International Journal of Computer Applications in 2014.

[21] “A Simple Algorithm for Testing Polygonal Chains for Simple Polygons” by R. E. Miller and J. W. Thatcher, published in Information Processing Letters in 1978.

致　　谢

路漫漫其修远兮，吾将上下而求索。行文至此，落笔为终。意味着四年大学生活也即将结束，我与山东大学的故事始于 2019 年金秋，终于 2023 年盛夏，回首四年光阴，转瞬即逝，收获满满，临近毕业之时，心里仍存感恩。

父母之爱子，则为之计深远。感谢你们含辛茹苦将我养育成人和二十一年来给予我在学业上的支持。希望时光走得慢些，你们平安幸福便是我最大的心愿。

桃李不言，下自成蹊。首先要感谢我的毕业论文指导老师——赵海森教授。赵海森教授治学严谨、知识渊博，从论文定题到撰写的每个步骤都离不开他给予的指导、认可和鼓励，让我在大学的学业生涯不留遗憾。愿您此后身体健康，万事顺心，工作顺利，事业有成；其次在此感谢大学有幸遇到的所有的老师，师恩难忘，我必会铭记于心。

日夜太长，岁月太短。感谢我的三个室友马慧颖、汤雨和钟欣怡，是你们陪伴了我一千多个日夜，我在和你们的相处中获得了许多正能量。祝我们毕业快乐，前程似锦。五湖四海来，天南地北去，愿我们江湖有梦，各自精彩。

致谢是终点，也是起点。感谢所有的遇见，也衷心祝愿母校山东大学满育英才。再见了滨海公路 72 号！山水有来路，早晚复相逢，愿此去经年，一路顺风，于顶峰相见。

附　　录

```
#include <iostream>
#include <GL/glut.h>
#include <math.h>
#include <vector>
#include <algorithm>

#define EPSILON 0.000001

using namespace std;

int g_iCurSelect = -1;

//鼠标左键状态
bool g_bLeft = false;

int g_iStartX = 0;
int g_iStartY = 0;

//记录窗体的宽高
int g_iWinW = 0;
int g_iWinH = 0;

struct MyPoint
{
    double x;
    double y;
};

vector<MyPoint> pt1 =
{
    {3 * 10,24 * 10 },
    {3.5 * 10,24 * 10 },
```

```
    {4.5 * 10,28 * 10 },
    {4 * 10,28 * 10 }
};

vector<MyPoint> pt2 =
{
    {3 * 20,20 * 10},
    {4 * 20,20 * 10},
    {5 * 20,28 * 10},
    {6 * 20,28 * 10},
    {6 * 20,28.1 * 10},
    {4.9 * 20,28.1 * 10},
    {3.9 * 20,20.1 * 10},
    {3 * 20,20.1 * 10},
};

vector<MyPoint> pt3 =
{
    {4 * 20,17 * 10},
    {5 * 20,17 * 10},
    {5 * 20,17.67 * 10},
    {7 * 20,19 * 10},
    {8.5 * 20,25 * 10},
    {8.5 * 20,27 * 10},
    {11 * 20,28 * 10},
    {11 * 20,28.1 * 10},
    {8.4 * 20,28.1 * 10},
    {8.4 * 20,25 * 10},
    {6.9 * 20,19.1 * 10},
    {3.8 * 20,17 * 10},
};

//后续要添加的内容 3
```

```
/*vector<MyPoint> pt3 =
{
    {-3,-24},
    {-100,-24},
    {-100,-90},
    {-50,-150},
    {-3,-90}
};*/

void RectMoveXY(vector<MyPoint>& pts, int iOffsetX, int iOffsetY)
{
    for (int i = 0; i < pts.size(); i++)
    {
        pts[i].x += iOffsetX;
        pts[i].y -= iOffsetY;
    }
}


void display_image(void)
{
    glLoadIdentity();
    glClearColor(0, 0, 0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glLineWidth(2);

    //glTranslatef(-300, -300, 0);
    //glRotatef(90, 0, 0, 1);

    //绘制顶点 1 的
    if (g_iCurSelect == 0)
```

```
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt1.size(); i++)
{
    glVertex2f(pt1[i].x, pt1[i].y);
}
glEnd();
glBegin(GL_POLYGON);
for (int i = 0; i < pt1.size(); i++)
{
    glVertex2f(-pt1[i].x, pt1[i].y);
}
glEnd();

//绘制顶点 2 的
if (g_iCurSelect == 1)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt2.size(); i++)
{
    glVertex2f(pt2[i].x, pt2[i].y);
```

```
}
glEnd();
glBegin(GL_POLYGON);
for (int i = 0; i < pt2.size(); i++)
{
    glVertex2f(-pt2[i].x, pt2[i].y);
}
glEnd();


//后续要添加的内容 3
if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt3.size(); i++)
{
    glVertex2f(pt3[i].x, pt3[i].y);
}
glEnd();

//后续要添加的内容 4
/*if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
```

```
        glColor3f(1.0, 0.0, 0.0);
    }
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt3.size(); i++)
    {
        glVertex2f(pt3[i].x, pt3[i].y);
    }
    glEnd();*/

    //后续要添加的内容 5
    /*if (g_iCurSelect == 2)
    {
        glColor3f(0.0, 1.0, 0.0);
    }
    else
    {
        glColor3f(1.0, 0.0, 0.0);
    }
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt3.size(); i++)
    {
        glVertex2f(pt3[i].x, pt3[i].y);
    }
    glEnd();*/

    //后续要添加的内容 6
    /*if (g_iCurSelect == 2)
    {
        glColor3f(0.0, 1.0, 0.0);
    }
    else
    {
        glColor3f(1.0, 0.0, 0.0);
```

```
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt3.size(); i++)
{
    glVertex2f(pt3[i].x, pt3[i].y);
}
glEnd();*/

//后续要添加的内容 7
/*if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt3.size(); i++)
{
    glVertex2f(pt3[i].x, pt3[i].y);
}
glEnd();*/

//后续要添加的内容 8
/*if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
```

```
        glBegin(GL_POLYGON);
        for (int i = 0; i < pt3.size(); i++)
        {
            glVertex2f(pt3[i].x, pt3[i].y);
        }
        glEnd();*/

    #if 0



        //cout << "p9" << endl;
        glBegin(GL_POLYGON);
        for (int i = 0; i < sizeof(pt9) / sizeof(MyPoint); i++)
        {
            glVertex2f(pt9[i].dX, pt9[i].dY);
            //cout << "x:" << pt9[i].dX << " y:" << pt9[i].dY << endl;
        }
        glEnd();
    #endif

        glFlush();
    }//display_image()

    //获取鼠标按下位置  Q1-b
    void mymouse(int button, int state, int x, int y)
    {
        int iReadX = x - g_iWinW / 2;
        int iReadY = g_iWinH / 2 - y;
        if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        {
            g_bLeft = true;
            std::cout << "GLUT_LEFT_BUTTON GLUT_DOWN " << "x:" <<
iReadX << " y:" << iReadY << endl;
```

```
g_iStartX = x;
g_iStartY = y;

MyPoint pt = { iReadX, iReadY };

if (PtInPolygon(pt, pt1))
{
    g_iCurSelect = 0;
}
else if (PtInPolygon(pt, pt2))
{
    g_iCurSelect = 1;
}
//后续要添加的内容 3
else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}
//4
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}*/
//5
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}*/
//6
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
```

```
        }*/
        //7
        /*else if (PtInPolygon(pt, pt3))
        {
            g_iCurSelect = 2;
        }*/
        //8
        /*else if (PtInPolygon(pt, pt3))
        {
            g_iCurSelect = 2;
        }*/

        else
        {
            g_iCurSelect = -1;
        }

        cout << "PtInPolygon g_iCurSelect:" << g_iCurSelect << endl;

    }
    else if (state == GLUT_UP)
    {
        g_bLeft = false;

    }

}


void MotionFunc(int iX, int iY)
{
    if (g_bLeft)
    {
```

```
int iOffSetx = iX - g_iStartX;
int iOffSety = iY - g_iStartY;

if (g_iCurSelect == 0)
{
    RectMoveXY(pt1, iOffSetx, iOffSety);
    glutPostRedisplay();
}
else if (g_iCurSelect == 1)
{
    RectMoveXY(pt2, iOffSetx, iOffSety);
    glutPostRedisplay();
}

//后续要添加的内容 3
else if (g_iCurSelect == 2)
{
    RectMoveXY(pt3, iOffSetx, iOffSety);
    glutPostRedisplay();
}
//后续要添加的内容 4
/*else if (g_iCurSelect == 2)
{
    RectMoveXY(pt3, iOffSetx, iOffSety);
    glutPostRedisplay();
}*/
//后续要添加的内容 5
/*else if (g_iCurSelect == 2)
{
    RectMoveXY(pt3, iOffSetx, iOffSety);
    glutPostRedisplay();
}*/
//后续要添加的内容 6
```

```
        /*else if (g_iCurSelect == 2)
        {
            RectMoveXY(pt3, iOffSetx, iOffSety);
            glutPostRedisplay();
        }*/
        //后续要添加的内容 7
        /*else if (g_iCurSelect == 2)
        {
            RectMoveXY(pt3, iOffSetx, iOffSety);
            glutPostRedisplay();
        }*/
        //后续要添加的内容 8
        /*else if (g_iCurSelect == 2)
        {
            RectMoveXY(pt3, iOffSetx, iOffSety);
            glutPostRedisplay();
        }*/



        g_iStartX = iX;
        g_iStartY = iY;

        std::cout << "MotionFunc" << iX << "    " << iY << endl;
    }
}

//Q2
void reShapesize(int w, int h)
{

#if 0
    //change the showing area, starting from the bottom left instead of default
```

```
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        //change resolution to w/h. Camera perspective so that the cube is showing on the screen.
        gluPerspective(45, (float)w / h, 4, 10);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //glTranslatef(0.0, 0.0, -6.0);
    #else
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        //设置平行视角以宽  高为面==
        glOrtho(-w / 2, w / 2, -h / 2, h / 2, 0, 1);

        glMatrixMode(GL_MODELVIEW);

        g_iWinW = w;
        g_iWinH = h;

    #endif

    }


    int main(int argc, char* argv[])
    {
        glutInit(&argc, argv);
        glutInitWindowSize(600, 600);

        glutCreateWindow("SIMPLE DISPLAY");
```

```
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);


    glShadeModel(GL_SMOOTH);
    glutMouseFunc(mymouse);
    glutDisplayFunc(display_image);
    glutReshapeFunc(reShapesize);

    glutMotionFunc(MotionFunc);
    //glutKeyboardFunc(keyboard);

    //glutTimerFunc(1000, timerFunc, 0);
    //
    /*glMatrixMode(GL_PROJECTION);
    glOrtho(0, global.w, 0, global.h, 0, 1);*/

    /*glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 600, 0, 600, 0, 1);*/

    glutMainLoop();
}
// project.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//
#pragma comment(lib,"glut32.lib")//opengl 库
#include "pch.h"
#include <iostream>
#include "glut.h"//opengl 头文件
#include"STLModel.h"
#include <math.h>
#include <vector>
```

```
#include <algorithm>

GLfloat m_tranlate[3]      = {0,0,-10};//用于平移，对应 X Y Z 平移量。按键 W:上    S:下    A:左  D:右
GLfloat m_rorate[2]        = {0,0}  ;//用于旋转，分别绕 X 轴和 Y 轴旋转的角度，用鼠标左键控制
GLfloat m_scale            = 1.0    ;//用于缩放，用鼠标中间滚轮控制
GLint   m_MouseDownPT[2] = {0,0}  ;//记录鼠标坐标点，用于控制旋转角度
bool    m_bMouseDown       = false  ;//记录鼠标左键是否按下，按下为 TRUE,初始值为 false

STLModel *model;
int m_model = 0;
//重新绘制
void display()
{
    glEnable(GL_LIGHT0);//启动一号灯
    glEnable(GL_LIGHTING);//开启光照
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // 擦除背景色和深度缓存
    glPushMatrix();//压栈
    glColor3f(1.0, 0, 0);//设置颜色为红色
    glTranslatef(m_tranlate[0], m_tranlate[1], m_tranlate[2]);//平移(X,Y,Z)
    glRotatef(m_rorate[0], 1, 0, 0);//旋转  绕 X 轴
    glRotatef(m_rorate[1], 0, 1, 0);//旋转  绕 Y 轴
    glScalef(m_scale, m_scale, m_scale);//缩放（X,Y,Z）
    //glutSolidTeapot(1.0);//绘制一个茶壶，1.0：指的是茶壶大小
    model->Draw(m_model);
    glPopMatrix();//出栈
    glutSwapBuffers();//交互前后缓冲区
```

```
}

//窗口大小改变事件
void ReshapeEvent(int width, int height)
{
    glViewport(0, 0, width, height);//视口在屏幕的大小位置
    glMatrixMode(GL_PROJECTION);// 投影矩阵
    glLoadIdentity();                    // 矩阵单位
    gluPerspective(45.0f, (GLfloat)width / (GLfloat)height, 0.1f, 10000.0f);// 设置投影矩阵
    glMatrixMode(GL_MODELVIEW);   // 模型矩阵
    glLoadIdentity();//单位矩阵化
    glEnable(GL_DEPTH_TEST);//启动深度检测
}
//空闲事件处理事件
void IdleEvent()
{
     glutPostRedisplay(); //刷新函数
}
//键盘事件 默认是英文输入法下的大写字母
void KeyboardEvent(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'W'://上移动
        m_tranlate[1] += 0.1;
        break;
    case 'S'://下移动
        m_tranlate[1] -= 0.1;
        break;
```

```
        case 'A'://左移动
            m_tranlate[0] -= 0.1;
            break;
        case 'D'://右移动
            m_tranlate[0] += 0.1;
            break;
        case 'Z':
            m_model = m_model == 0 ? 1 : 0;
            break;
        }
    }
    //鼠标事件
    void MouseEvent(int button, int state, int x, int y)
    {

        //1、鼠标缩放：往上滚动放大；往下滚动缩小
        if (state == GLUT_UP && button == GLUT_WHEEL_UP)//滚轮往上滚动
        {
            m_scale += 0.1;
        }

        else if (state == GLUT_UP && button == GLUT_WHEEL_DOWN)//滚轮往
下滚动
        {
            m_scale -= 0.1;
            if (m_scale < 0.1) m_scale = 0.1;
        }


        //【2 鼠标左键是否按下】
        if (state == GLUT_DOWN && button == GLUT_LEFT_BUTTON)
        {
```

```
            m_bMouseDown = true;//鼠标左键按下
            m_MouseDownPT[0] = x;//记录当前 X 坐标
            m_MouseDownPT[1] = y;////记录当前 Y 坐标
        }
        else
        {
            m_bMouseDown = false;//鼠标左键弹起，结束旋转
    }
    //按下鼠标按钮移动鼠标事件
    void MotionEvent(int x, int y)
    {
        if (m_bMouseDown)//如果鼠标左键被按下
        {
            m_rorate[0] += y - m_MouseDownPT[1];//通过滑动鼠标改变旋转的角
度
            m_rorate[1] += x - m_MouseDownPT[0];//通过滑动鼠标改变旋转的角
度

            m_MouseDownPT[0] = x;//记录当前 X 坐标
            m_MouseDownPT[1] = y;//记录当前 Y 坐标
        }
    }
    //检测鼠标进入或离开窗口
    void MouseEntry(int state)
    {
        /*state 有两个值表明是离开还是进入窗口:
        GLUT_LEFT
        GLUT_ENTERED*/
    }
    //点击菜单响应事件
    void MenuEvent(int choose)
```

```
{
    switch (choose)
    {
    case 1://复位：把旋转平移缩放的值复位
        //用于平移，对应 X Y Z 平移量。按键 W:上  S:下    A:左  D：右
        m_tranlate[0] = 0;
        m_tranlate[1] = 0;
        m_tranlate[2] = -10;

        //用于旋转，分别是绕 X 轴 和 Y 轴旋转的角度，用鼠标左键控制
        m_rorate[0] = 0;
        m_rorate[1] = 0;

        //用于缩放，用鼠标中间滚轮控制
        m_scale = 1.0;

        //记录鼠标坐标点，用于控制旋转角度；
        m_MouseDownPT[0] = 0;
        m_MouseDownPT[1] = 0;

        //记录鼠标左键是否按下，按下为 true,初始值为 false
        m_bMouseDown = false;
        break;
    case 2://功能待定：暂时不做处理
        break;

    }
}

#define EPSILON 0.000001

using namespace std;
```

```
int g_iCurSelect = -1;

//鼠标左键状态
bool g_bLeft = false;

int g_iStartX = 0;
int g_iStartY = 0;

//记录窗体的宽高
int g_iWinW = 0;
int g_iWinH = 0;

struct MyPoint
{
    double x;
    double y;
};

vector<MyPoint> pt1 =
{
    {3 * 10,24 * 10 },
    {3.5 * 10,24 * 10 },
    {4.5 * 10,28 * 10 },
    {4 * 10,28 * 10 }
};

vector<MyPoint> pt2 =
{
    {3 * 20,20 * 10},
    {4 * 20,20 * 10},
    {5 * 20,28 * 10},
    {6 * 20,28 * 10},
```

```
    {6 * 20,28.1 * 10},
    {4.9 * 20,28.1 * 10},
    {3.9 * 20,20.1 * 10},
    {3 * 20,20.1 * 10},
};

vector<MyPoint> pt3 =
{
    {4 * 20,17 * 10},
    {5 * 20,17 * 10},
    {5 * 20,17.67 * 10},
    {7 * 20,19 * 10},
    {8.5 * 20,25 * 10},
    {8.5 * 20,27 * 10},
    {11 * 20,28 * 10},
    {11 * 20,28.1 * 10},
    {8.4 * 20,28.1 * 10},
    {8.4 * 20,25 * 10},
    {6.9 * 20,19.1 * 10},
    {3.8 * 20,17 * 10},
};

//后续要添加的内容 3
/*vector<MyPoint> pt3 =
{
    {-3,-24},
    {-100,-24},
    {-100,-90},
    {-50,-150},
    {-3,-90}
};*/

void RectMoveXY(vector<MyPoint>& pts, int iOffsetX, int iOffsetY)
```

```
{
    for (int i = 0; i < pts.size(); i++)
    {
        pts[i].x += iOffsetX;
        pts[i].y -= iOffsetY;
    }
}


void display_image(void)
{
    glLoadIdentity();
    glClearColor(0, 0, 0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glLineWidth(2);

    //glTranslatef(-300, -300, 0);
    //glRotatef(90, 0, 0, 1);

    //绘制顶点 1 的
    if (g_iCurSelect == 0)
    {
        glColor3f(0.0, 1.0, 0.0);
    }
    else
    {
        glColor3f(1.0, 0.0, 0.0);
    }
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt1.size(); i++)
    {
```

```
        glVertex2f(pt1[i].x, pt1[i].y);
    }
    glEnd();
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt1.size(); i++)
    {
        glVertex2f(-pt1[i].x, pt1[i].y);
    }
    glEnd();

    //绘制顶点 2 的
    if (g_iCurSelect == 1)
    {
        glColor3f(0.0, 1.0, 0.0);
    }
    else
    {
        glColor3f(1.0, 0.0, 0.0);
    }
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt2.size(); i++)
    {
        glVertex2f(pt2[i].x, pt2[i].y);
    }
    glEnd();
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt2.size(); i++)
    {
        glVertex2f(-pt2[i].x, pt2[i].y);
    }
    glEnd();
```

```
//后续要添加的内容 3
if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt3.size(); i++)
{
    glVertex2f(pt3[i].x, pt3[i].y);
}
glEnd();

//后续要添加的内容 4
/*if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt3.size(); i++)
{
    glVertex2f(pt3[i].x, pt3[i].y);
}
glEnd();*/

//后续要添加的内容 5
```

```
/*if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt3.size(); i++)
{
    glVertex2f(pt3[i].x, pt3[i].y);
}
glEnd();*/

//后续要添加的内容 6
/*if (g_iCurSelect == 2)
{
    glColor3f(0.0, 1.0, 0.0);
}
else
{
    glColor3f(1.0, 0.0, 0.0);
}
glBegin(GL_POLYGON);
for (int i = 0; i < pt3.size(); i++)
{
    glVertex2f(pt3[i].x, pt3[i].y);
}
glEnd();*/

//后续要添加的内容 7
/*if (g_iCurSelect == 2)
```

```
    {
        glColor3f(0.0, 1.0, 0.0);
    }
    else
    {
        glColor3f(1.0, 0.0, 0.0);
    }
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt3.size(); i++)
    {
        glVertex2f(pt3[i].x, pt3[i].y);
    }
    glEnd();*/

    //后续要添加的内容 8
    /*if (g_iCurSelect == 2)
    {
        glColor3f(0.0, 1.0, 0.0);
    }
    else
    {
        glColor3f(1.0, 0.0, 0.0);
    }
    glBegin(GL_POLYGON);
    for (int i = 0; i < pt3.size(); i++)
    {
        glVertex2f(pt3[i].x, pt3[i].y);
    }
    glEnd();*/

#if 0
```

```
        //cout << "p9" << endl;
        glBegin(GL_POLYGON);
        for (int i = 0; i < sizeof(pt9) / sizeof(MyPoint); i++)
        {
            glVertex2f(pt9[i].dX, pt9[i].dY);
            //cout << "x:" << pt9[i].dX << " y:" << pt9[i].dY << endl;
        }
        glEnd();
    #endif

        glFlush();
    }//display_image()

    //获取鼠标按下位置  Q1-b
    void mymouse(int button, int state, int x, int y)
    {
        int iReadX = x - g_iWinW / 2;
        int iReadY = g_iWinH / 2 - y;
        if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        {
            g_bLeft = true;
            std::cout << "GLUT_LEFT_BUTTON GLUT_DOWN " << "x:" <<
iReadX << " y:" << iReadY << endl;

            g_iStartX = x;
            g_iStartY = y;

            MyPoint pt = { iReadX, iReadY };

            if (PtInPolygon(pt, pt1))
            {
                g_iCurSelect = 0;
            }
```

```
else if (PtInPolygon(pt, pt2))
{
    g_iCurSelect = 1;
}
//后续要添加的内容 3
else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}
//4
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}*/
//5
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}*/
//6
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}*/
//7
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
}*/
//8
/*else if (PtInPolygon(pt, pt3))
{
    g_iCurSelect = 2;
```

```
        }*/

        else
        {
            g_iCurSelect = -1;
        }

        cout << "PtInPolygon g_iCurSelect:" << g_iCurSelect << endl;

    }
    else if (state == GLUT_UP)
    {
        g_bLeft = false;

    }

}


void MotionFunc(int iX, int iY)
{
    if (g_bLeft)
    {
        int iOffSetx = iX - g_iStartX;
        int iOffSety = iY - g_iStartY;

        if (g_iCurSelect == 0)
        {
            RectMoveXY(pt1, iOffSetx, iOffSety);
            glutPostRedisplay();
        }
        else if (g_iCurSelect == 1)
        {
```

```
        RectMoveXY(pt2, iOffSetx, iOffSety);
        glutPostRedisplay();
    }

    //后续要添加的内容 3
    else if (g_iCurSelect == 2)
    {
        RectMoveXY(pt3, iOffSetx, iOffSety);
        glutPostRedisplay();
    }
    //后续要添加的内容 4
    /*else if (g_iCurSelect == 2)
    {
        RectMoveXY(pt3, iOffSetx, iOffSety);
        glutPostRedisplay();
    }*/
    //后续要添加的内容 5
    /*else if (g_iCurSelect == 2)
    {
        RectMoveXY(pt3, iOffSetx, iOffSety);
        glutPostRedisplay();
    }*/
    //后续要添加的内容 6
    /*else if (g_iCurSelect == 2)
    {
        RectMoveXY(pt3, iOffSetx, iOffSety);
        glutPostRedisplay();
    }*/
    //后续要添加的内容 7
    /*else if (g_iCurSelect == 2)
    {
        RectMoveXY(pt3, iOffSetx, iOffSety);
```

```
                glutPostRedisplay();
            }*/
            //后续要添加的内容 8
            /*else if (g_iCurSelect == 2)
            {
                RectMoveXY(pt3, iOffSetx, iOffSety);
                glutPostRedisplay();
            }*/



            g_iStartX = iX;
            g_iStartY = iY;

            std::cout << "MotionFunc" << iX << "    " << iY << endl;
        }
    }

    //Q2
    void reShapesize(int w, int h)
    {

    #if 0
        //change the showing area, starting from the bottom left instead of default
        glViewport(0, 0, (GLsizei)w, (GLsizei)h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        //change resolution to w/h. Camera perspective so that the cube is showing on
the screen.
        gluPerspective(45, (float)w / h, 4, 10);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //glTranslatef(0.0, 0.0, -6.0);
```

```
#else
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    //设置平行视角以宽 高为面==
    glOrtho(-w / 2, w / 2, -h / 2, h / 2, 0, 1);

    glMatrixMode(GL_MODELVIEW);

    g_iWinW = w;
    g_iWinH = h;

#endif

}


int main(int argc, char** argv)
{
    model    = new STLModel("res/new.stl");
    glutInit(&argc, (char**)argv); //对 GLUT 进行初始化
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH); //设置模式为：双缓冲，深度缓存区
    glutInitWindowPosition(110, 120);//窗口位置
    glutInitWindowSize(600, 600);//窗口大小
    glutCreateWindow("project");   //创建窗口
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutDisplayFunc(display);       // 重新绘制事件
    //-------------注册回调函数----------------------//
    glutKeyboardFunc(KeyboardEvent); // 键盘事件
    glutMouseFunc(MouseEvent);            //鼠标事件
```

```
    glutReshapeFunc(ReshapeEvent);     //窗口大小发生变化事件
    glutMotionFunc(MotionEvent);       //鼠标移动事件
    glutIdleFunc(IdleEvent);           //空闲处理事件
    glutEntryFunc(MouseEntry);         //检测鼠标进入或离开窗口
    {
    glutCreateMenu(MenuEvent);         //创建菜单
    glutAddMenuEntry("复位", 1);       //菜单项 1
    glutAddMenuEntry("待定", 2);       //菜单项 2
    glutAttachMenu(GLUT_RIGHT_BUTTON);//鼠标右键按下弹出菜单
    }
    glutMainLoop();                    //调用已注册的回调函数
    return 0;
}
#pragma once

#include<vector> //包括容器

typedef float M3DVector3f[3];//定义坐标点
class STLModel
{
public:
    STLModel();//构造函数
    STLModel(const char* stlFileName);//构造函数
    ~STLModel();//析构函数
    bool LoadStlFile(const char* stlFileName);//加载 STL 模型文件并解析
    void Draw(int model);//绘制模型

protected:
    struct RenderTri//一个三角形所有数据结构体
    {
        M3DVector3f m_Normal; //法线
```

```
        M3DVector3f m_Vertx1;//三角形第一个顶点
    };
private:
    std::vector<RenderTri> m_MeshTris;      //保存加载的三角形数据
    M3DVector3f m_MaxPos;//记录最大的 X Y Z 点  方便平移 缩放用来最佳显示
    M3DVector3f m_MinPos;//记录最小的 X Y Z 点 方便平移 缩放用来最佳显示
};
#include "STLModel.h"
#include "glut.h"//opengl 头文件
bool STLModel::LoadStlFile(const char* stlFileName)
{
    FILE* file = NULL;
    errno_t err;//记录错误
    int Triangle_num = 0;//记录读取的三角形个数
    err = fopen_s(&file, stlFileName, "r");//打开文件
    if (err != 0)
        return false;//如果读取文件错误，返回

    char buffer[133];//每行字符串
    char str1[80], str2[80];//提取的字符串
    int lineno = 0;//记录读取的行数


    //读取头文件  solid finename stl    ：  文件路径及文件名
    fgets(buffer, 133, file);
    lineno++;
    sscanf_s(buffer, "%s %s", str1, 80, str2, 80);
    assert(_strcmpi(str1, "solid") == 0);
    if (_strcmpi(str1, "solid") != 0)//检测是否有 solid ,如果没有返回错误
```

```
        return false;

    m_MeshTris.clear();//清空数据


    while (fgets(buffer, 133, file))//开始读取每一个三角形数据
    {
        lineno++;//记录读取的行数  加 1

        RenderTri tri;//三角形结构体数据，包括 3 个顶点坐标和 1 个法线坐标

        //读取法线坐标    facet normal x y   z   ：   三角片面法向量的 3 个分量
        sscanf_s(buffer, "%s %s %f %f %f", str1, 80, str2, 80, &(tri.m_Normal[0]),
&(tri.m_Normal[1]), &(tri.m_Normal[2]));
        if (_strcmpi(str1, "facet") != 0 || _strcmpi(str2, "normal") != 0)
            continue;
        fgets(buffer, 133, file);
        lineno++;
         // 过滤掉  outer loop
        sscanf_s(buffer, "%*s %*s");
        fgets(buffer, 133, file);
        lineno++;
        //读取第一个顶点坐标
        sscanf_s(buffer, "%s %f %f %f", str1, 80, &(tri.m_Vertx1[0]),
&(tri.m_Vertx1[1]), &(tri.m_Vertx1[2]));
        fgets(buffer, 133, file);
        lineno++;
        fgets(buffer, 133, file);
        lineno++;
        sscanf_s(buffer, "%*s");

        //过滤掉       endfacet
```

```
            fgets(buffer, 133, file);
            lineno++;
            sscanf_s(buffer, "%*s");
            //记录最大值和最小值，方便平移和缩放 让模型在视图正中间显示
            {
                //记录坐标点
                if (Triangle_num == 0)
                {
                    for (int i = 0; i < 3; i++)
                        m_MaxPos[i] = m_MinPos[i] = tri.m_Vertx1[i];//赋初始值

                }
                Triangle_num += 1;//三角形个数加 1
                for (int i = 0; i < 3; i++)
                {
                    //比较 并 记录最大值
                    if (m_MaxPos[i] < tri.m_Vertx1[i])m_MaxPos[i] =
tri.m_Vertx1[i];
                    //比较 并 记录最小值
                    if (m_MinPos[i] > tri.m_Vertx1[i])m_MinPos[i] =
tri.m_Vertx1[i];
                }
            }

            std::cout << "第" << Triangle_num << "个三角形:" << std::endl;//显示三
角形个数
            m_MeshTris.push_back(tri);//把三角形数据保存到容器中
        }

        fclose(file);//关闭文件
        return true;
    }
```

```
void STLModel::Draw(int model)//绘制模型
{
    glEnable(GL_NORMALIZE);//重新计算法线坐标
    glPushMatrix();//压栈
    float    m_scale =100;//缩放倍数
    m_scale = m_scale < (1.0 / (m_MaxPos[0] - m_MinPos[0])) ? m_scale : (1.0 /
(m_MaxPos[0] - m_MinPos[0]));//比较长缩放倍数 ， 记录小的倍数值
    glScalef(m_scale,m_scale,m_scale);//缩放  让长宽高最大是  1
    glTranslatef(-(m_MinPos[0] + m_MaxPos[0]) / 2.0, -(m_MinPos[1] +
m_MaxPos[1]) / 2.0, -(m_MinPos[2] + m_MaxPos[2]) / 2.0);//平移，让模型中心在原
点
    glBegin(GL_TRIANGLES);//绘制三角形
    if (model == 0)
    {
        for (size_t i = 0; i < m_MeshTris.size(); i++)//循环所有三角形
        {
            //第一个点数据
            glNormal3f(m_MeshTris[i].m_Normal[0],
m_MeshTris[i].m_Normal[1], m_MeshTris[i].m_Normal[2]);
            glVertex3f(m_MeshTris[i].m_Vertx1[0], m_MeshTris[i].m_Vertx1[1],
m_MeshTris[i].m_Vertx1[2]);
    }
    glEnd();//结束绘制
    glPopMatrix();//出栈
}
```

译文中文

利用混合进化算法求解二维装箱问题

克里斯蒂安·布鲁姆·韦雷娜·施密德

摘要

处理二维箱包装的组合优化问题的应用，例如，在运输/仓储和玻璃、木材和金属的切割。在本工作中，我们考虑了在自由断头台切割下的定向二维箱子包装问题，其中给出了一组定向矩形项目，它必须被包装到最小数量的相同大小的箱子中。我们的算法建议来解决这个问题涉及一个进化算法，它大量使用随机的一次启发式来构造解决方案。并将该算法的计算结果与文献中的一些最佳方法进行了比较。这种比较表明，我们的算法与最先进的方法非常有竞争力。特别地，找到了四个以前未解决的实例的最优解。

关键词：二维装箱，自由断头台切割，进化算法，混合计算系统

1.介绍

在这项工作中，我们提出了一种混合进化算法来解决二维箱包装问题，其中项目有一个五个固定的方向和自由断头台切割是允许的。根据洛迪等人的说法。[1]这个问题在文献中被表示为 2BP|O|[1]。今后，我们将仅用 2BP 来讨论这个问题。关于复杂性，请注意，2BP 问题是一个 np 硬的组合优化问题（见[2]）。一般来说，垃圾箱包装问题在过去已经得到了广泛的研究。它们受欢迎的主要原因是大量的真实应用程序。2BP 的实际世界应用包括，例如，在运输或仓储的环境中切割玻璃、木材或金属和包装（见[3,4]）。如上所述，在这项工作中，我们使用了一种混合进化算法来处理 2BP。该算法可以归因于混合元启发式，这是一个研究领域，处理结合元启发式，如进化算法和标签搜索，与其他优化技术，如启发式和精确方法。它已经表明，这种方法可能强烈有利于联合算法方法之间的协同作用。有关混合元启发式的更多信息，我们请向感兴趣的读者咨询[5]。本文中提出的进化算法的混合方面涉及到解的表示。事实上，该算法的搜索空间并不是直接研究问题的解决方案，而是包含了属于一个问题实例的条目的所有排列。为了确定一个排列的目标函数值，采用了[6]中提出的 LGFi 启发式方法。该启发式将所有项的任

何排列作为输入，并以特定的方式为相应的 2BP 问题实例生成相应的解决方案。实验结果表明，该算法优于现有的算法方法。事实上，将所有考虑的 500 个问题实例所使用的箱子数加起来，进化算法达到 7239，这是为该问题提出的任何算法所达到的最佳值。本论文的组织结构如下。在第 2 节中，我们提供了相关工作的概述。接下来，在第 3 节中，我们将提供对所解决的问题的技术描述。所提出的算法将在第 4 节中提出。最后，第 5 节提供了实验评价，第 6 节给出了结论和未来展望。

2.相关工作

下面我们将关注最近关于 2BP 问题的（元）启发式的工作。在[7,8,9,10]中可以找到关于早期工作的一个很好的概述。

启发式。文献主要区分了一阶段方法和两阶段方法。非相算法将项目直接打包到箱子中，而两阶段算法首先将项目打包到级别中，即，不超过箱子宽度的项目的水平包装，然后将这些级别堆叠到箱子中。众所周知的水平填充算法是下拟合降低高度（NFDH）、第一拟合降低高度（FFDH）和最佳拟合降低高度（BFDH）[11]。这些策略最初是针对一维料箱包装问题开发的，但后来又适用于条带包装问题和二维情况。这三种启发式都要求项目按非增加的高度排序，高度表示它们打包的顺序。此外，他们还把这些物品装进一个高度有限的箱子里。

其中最重要的单阶段方法包括交替方向（AD）[1]、左下填充（BLF）[12]、改进的最低间隙填充（LGFi）[6]和接触周长（TP）[1]。关于两阶段方法，我们区分了基于 NFDH 的混合下一步匹配（HNF）（见[13]），基于 FFDH 的混合第一匹配（HFF）[14]和有限最佳带（FBS）[15]，有时也被称为混合最佳匹配，它基于 BFDH。另一个例子是关于背包包装（KP）[1]。最后，地板天花板（FC）[1]可以被视为对 FBS 的一种改进。目前可用的 2BP（标记为 SCH）的最佳启发式是基于通过列生成的方法求解问题[16]的集合覆盖公式。在五个阶段，通过使用贪婪程序和文献中的快速构造启发式算法生成所有可能列的一个相当小的子集。在第二阶段，通过基于拉格朗日的启发式来求解结果集覆盖实例。此外，一些为三维包装开发的启发式方法有时可以很容易地应用于 2BP。一个例子是基于[17]的极点启发式。这个启发式使用极端点来确定箱子中可以放置物品的所有点。极端点可以是已放置的项目的角，也可以是由已放置项目的扩展边生成的点。每次将一件

物品放入垃圾桶时，这些点都会被更新。对于放置这些项目，我们使用了一个修改版本的 BFDH。

元启发式。除了启发式之外，最近的研究者也探索了元启发式在 2BP 问题上的应用。最早为 2BP 开发的元启发式是表搜索（TS）[18,7]。此外，其他研究人员还应用了引导局部搜索（GLS）[19]和权重退火（WA）[20]。在[21]中提出了一种基于贪婪启发式的相当简单的元启发式，标记为 HBP。HBP 为每个项目分配一个分数。然后，为了构建一个解决方案，根据分数的非递增值来考虑这些项目。在构建一个解决方案后，使用一定的标准更新分数。重复此过程，直到满足预先定义的停止标准。最后，目前表现最好的元启发式算法是贪婪随机自适应搜索过程（GRASP）和变量邻域下降（VND）[22]之间的混合体。掌握的解决构建阶段是基于集装箱装载五域的最大空间启发式。

3．2d 料箱包装问题

下面通过一个 ILP 模型对所考虑的问题进行了技术描述

从[23]。1 在这个模型中，让 I = {1，...，n}是一组必须打包成最小数量的相同大小的箱子。每个项目∈以其宽度和高度∈为特征。在本文的背景下，注意 W 和 H 以及 wi 和 hi（对于所有∈）是整数值，这足以解决所考虑的问题实例。ILP 模型由以下变量组成。对于每个 i∈I

1. 变量 mi 的值表示放置项目 i 的箱子的数量；

2. 变量 xi 和 yi 的值表示 bin mi 中项目 i 位置的左下坐标.

此外，对于每一对 i，j∈i（这样 i j）

1. 二进制变量 li j 被设置为 1i ffff 项 i 位于项目 j 的左侧；

2. 二进制变量 bi j 被设置为 1i ffff 项 i 位于项 j 的下面；

3. 二进制变量 pi j 假设值为 1i ffff mi < mj.

最后，变量 v 表示正在使用的箱子的数量。使用这些变量，以下 ILP 解决了没有项目旋转和自由断头台切割。

在此，约束(2)、(3)、(4)、(6)和(7)注意两个项目之间和/或项目与箱子的边界之间没有重叠。约束条件(5)负责对 pij 变量的正确设置。此外，约束条件(8)和(9)将 bin 编号限制为[1，v]。最后，约束（10）是对称破缺约束，便于 MIP 求解器工作的对称破缺约束。

4.提出的算法

正如在引言中已经提到的，所提出的进化算法是强烈地基于启发式 LGFi，这是由黄和李在[6]中描述的。在下面，我们首先概述了 LGFi 的使用方式启发式工作。然后，详细描述了该进化算法。

1 请注意，在[24].902 中提出了一个替代模型

克里斯蒂安·布鲁姆和计算机计算机科学 18（2013）899-908

算法 12BP（EA-LGFi）

1: 输入：参数 psize，板条箱， κ 和 δ 的值

2: P := GenerateInitialPopulation (psize, κ)

3: 而停止标准不满足时做

4:

P：=交叉（P、板条箱、 δ ）

5: P := AddNewSolutions (P , psize, κ)

6: 结束时

7: 输出：已找到的最佳解决方案

4.1.启发式 LGFi

请注意，LGFi 是一个可分为两阶段的启发式算法。在预处理阶段，项目被排序到一个列表中，而在

包装阶段，这些物品从清单中包装到箱子里。更具体地说，在预处理阶段，项目按非增加区域排序，作为五种标准。关系被高度和宽度之间的绝对增量所打破。包装阶段是一个迭代过程，其中执行以下操作

在每次迭代中都要执行。首先，确定可以放置物品的最左下方的位置。这个位置从此就被称为当前的位置。然后，计算出关于这个位置的两个间隙。水平间隙定义为当前位置与箱子的右边框或第一个项的左边缘与当前位置与箱子的右边框之间的距离。之间的距离

箱子的当前位置和上边框定义了垂直间隙的值。较小的间隙的值称为电流间隙。如果水平间隙是当前间隙，则将当前间隙与未打包项目列表中项目的宽度进行比较；如果垂直间隙是当前间隙，则将与未打包项目列表中项目的高度进行比

较。完全填充间隙的五个项目被放置在左下角的标识位置。如果不存在这样的项目，则将没有重叠的五项与它放在底部

在当前位置的左转角。如果也不存在这样的项目，则必须将部分区域声明为浪费区域，其工作原理如下。创建具有水平间隙宽度的损耗区域。损耗区域的高度被选择为最低相邻项目的上边缘的高度，如果没有相邻项目存在，则被选择为箱子的高度。最后，如果找不到当前位置，如果存在未包装的项目，则打开一个新的箱子。

4.2.进化算法

在该算法的背景下，一个从此标记为 ea-LGFi 的解决方案是 LGFi 的输入序列 s。请注意，任何输入序列 s 都是必须打包的所有项的排列。该排列的 j 位置的项（其中 j = 1，...，n）用 sj 表示。将 LGFi 应用于 s，求出解 s 的目标函数值 f (s)。EA-LGFi 的伪代码显示在 Alg 中。 1.EALGFi 的第五步是生成大小为 psize 的初始种群（参见函数生成初始种群（psize， κ ））。

然后，在每次迭代中，在函数交叉（P，δ）中应用交叉算子，生成板条箱 •psize 新解，其中 0<板条箱≤1。这导致了一个新的种群 P 小于 p 大小的解。缺失的 psize −|P|解决方案是由函数 AddNew 解决方案（P，psize， κ ）生成的。下面将更详细地概述算法 EA-LGFi 的三个功能。

（psize， κ ）在这个函数中，psize 解的概率生成如下。首先，让我们表示输入序列，其中的项目按不递增的区域排序，作为确定性的输入序列。一个项目 i 在确定性输入序列中的位置从此被称为 posi。初始种群的任何解 s 都是基于确定性输入的概率生成的序列这样做如下。首先，请记住，项目的总数用 n 表示。然后以以下方式为每个项 i 分配一个值 vi：*vi := (n − posi)* $^{\kappa}$ (11)

其中， κ ≥1 是该函数的输入参数之一。s 的位置从 1 到 1 之间被填充以迭代的方式。在每一步中，让 I⊆I 是尚未分配给 s 的项集. *一个项目 i ∈I 是根据 903 克里斯蒂安布鲁姆和维雷纳施密德/普罗迪亚计算机科学 18（2013）899-908 的概率 p（i 比例于 vi：|I）（为所有 i ∈I）通过轮盘赌轮选择。计算了这些概率 p（i|I）*p (i|I) = vi i∈I vi (12)

请注意，参数 κ 的值越大，新生成的输入序列 s 与确定性输入序列的值就越相似.交叉（P，板条箱， δ ）该算子受到基于均匀顺序的交叉[25]的启发，对 P 的每

个最佳板条箱·|P|解进行重组，其中为 0<板箱≤1 是该算法的一个参数。广泛的经验检验表明，一个交叉率箱=0。7 最适合于手头的实例。对于 P 的最佳箱·|P|解决方案集合中的每个解决方案，从 P 中选择一个交叉伙伴 sc∈P（即 sc s）通过轮盘赌和轮轮的选择。假定 P 是一个有序的列表，其中的解被排序以一种非增加的方式增加到它们的目标函数值。连接被最后一个箱子的负载打破，也就是说，在最后一个箱子中具有较低负载的解决方案被预先排序。设 pos (s)表示解 s 在 p 中的位置，其中 δ ≥1 是算法的一个参数。给定两个跨界合作伙伴 s 和 sc，一个 o ffff 弹簧溶液，所以 ffff 是生成的方法如下所述。第一，三个指针（k、l 和 r)被初始化为第 5 个位置。所以 ffff 的 n 个位置从 1 到 n 个位置被填充如下。如果 sk = sc 那么，那么 ffff r ：= sk。换句话说，如果解 s 的位置 k 和解 sc 的位置 l 包含相同的项，则该项被放置在的位置 ro ffff 弹簧溶液，所以 ffff。接下来，位置指针 r 递增，位置指针 k 和 l 向右移动，直到到达最近的位置，其中包含一个项目尚未出现在解决方案中，所以 ffff。在 sk sc l 的情况下，解的 r 位置使 ffff 在 sk 和 sc 中概率选择 l，其中的概率为 0。75 是指源自这两个解决方案中较好的一个方案的项目。然后，位置指针 r 被递增。此外，选择项目的解决方案的位置指针向右移动，直到到达包含尚未在解决方案所以 ffff 中出现的项目的最近位置。通过使用 ffff 作为 LGFi 的输入来评估所得到的解决方案。如果 f（so ffff）<f(s)或 f（so ffff）=f(s)，那么 ffff 比最后一个箱子中的负载低，解将 ffff 添加到新的总体 P 中，否则解 s 添加到 P 中. 该函数（P，psize，κ）概率地生成 psize−|P |解决方案，方法与函数生成初始填充（psize，κ）相同。

5.实验评价

EA-LGFi 在 ANSI C++中实现，使用 GCC 4.4 进行编译软件。我们概述的结果是在 AMD64X2 4400 处理器和 4g 内存的 PC 上获得的。将该算法应用于文献中包含 500 个问题实例的基准测试集。在对算法行为的初步研究之后，给出了详细的实验评价。

5.1.文献中提供了 2BP 的 10 类问题实例。

Berkey 和 Wang 在[15]中提出了一个包含六个类（I-VI）的五个实例集。对于这些类别，项目的宽度和高度都是从表 1 中所示的间隔中均匀随机选择的。此外，在箱子的宽度(W)和高度(H)的类别。实例大小取自{20、40、60、80、100}。Berkey

和 Wang 为具有实例大小的类的每个组合提供了 10 个实例。这将导致总共有 300 个问题实例。

2 这 500 个实例可以从 http://www.or.deis.unibo.it/research.html 下载。904

克里斯蒂安·布鲁姆和计算机计算机科学 18（2013）899-908

表 1.指定的实例类别 I-VI（由[15]提供）。

第二个实例集，由类 VII-X 组成，是由马泰罗和维戈在[26]中引入的。一般来说，他们考虑了四种不同类型的项目，如表 2 所示。四种项目类型在项目的宽度 ffff 和高度 ffff 的限制。然后，基于这四种项目类型，马泰罗和维戈介绍了四种实例，区分每种类型中包含的项目的百分比。作为一个例子，让我们考虑第七类的一个实例。这种实例的 70%的项目是类型 1,10%的项目是类型 2，另外 10%的项目是类型 3，其余 10%的项目是类型 4。这些百分比在表 3 中给出。与第一个实例集的情况一样，实例大小取自{20、40、60、80、100}.

由 Martello 和 Vigo 设置的实例由每个具有实例大小的类组合的 10 个实例组成。这将导致总共有 200 个问题实例。

表 2。类 VII-X 的项目类型（如[26]中所介绍的）。请注意，对于所有的类，它都持有 W = H = 100。

表 3.指定实例类别 VII-X（由[26]提供）。

类类型 1 类型 2 类型 3 类型 4

VII 70% 10% 10% 10%

八、二、10% 70% 10% 10%

IX 10% 10% 70% 10%

X 10% 10% 10% 70%

5.2.调优实验

首先，我们选择了 5 数量的解决方案评估作为 EA-LGFi 运行的停止标准。特别是，在初始实验之后，我们为每次 EA-LGFi 的运行选择了 106 个解决方案评估的预算。初始实验后，参数 κ 的数值设为 10。选择要进行调整的参数是种群大小 psize 和 δ 。请记住， δ 的值用于计算解被选择为交叉伙伴的概率。更具体地说，我们考虑了 psize∈{10,100}和 δ ∈{1,5,10,15,20}。一般来说， δ 的值越高，在选择交叉伙伴 sc 时，就越多，好的解决方案就越差。然后，EA-LGFi 对 δ 和 psize 的每个组合分别应用于 500 个问题实例。在为所有 500 个实例生成的最佳解决方

案中，在每个参数值组合中使用的箱子数的总和如图 1 所示。尽管在算法性能上的双重误差相当小，但较高的 δ 值似乎比较小的 δ 值效果更好。此外，10 的人口规模似乎比 905 稍好一些。

图 1。调整 EA-LGFi 的结果。

人口规模为 100 人。在下一节中给出的 EA-LGFi 的五个最终结果是用

$\delta = 20$ 和 psize = 10。

5.3.数值结果

下面将 EA-LGFi 的结果与目前可用于 2BP 问题的两种最佳算法进行了比较：[16]的覆盖启发式（SCH）集和[22]的混合抓取方法。

数字计算结果如表 4 所示，这是针对 2BP 问题的传统方法。每个表行显示实例类（I-X）和项目数量（20-100）之间组合的 10 个实例的结果。关于这些列，一个标记的 LB 包含了从文献中已知的 10 个相应问题实例的最佳下界值的和。比较算法的结果每个算法两列。第一列（带有标题值）提供了为 10 个相应问题实例生成的最佳解决方案中使用的箱子数量的总和。例如，算法 SCH 为第 I 类的 10 个实例（20 个项目）生成的最佳解决方案总共占 71 个箱子。如果一个值对应于任何算法获得的最佳性能，则用粗体突出显示。此外，在 EA-LGFi 的情况下，如果一个值比目前最好的知道值要好，则用星号标记。第二列（标题时间(s)）显示了平均计算时间（秒），以找到 10 个实例类和项目数量之间组合的问题实例的最佳解决方案。例如，算法 SCH 平均需要 0.06 秒来找到 10 个第 I 类实例（20 个项目）的最佳解决方案。最后，表 4 的最后一行提供了所有 500 个问题实例的结果的摘要。对每个算法，都给出了所使用的箱子数的总和，以及 500 个实例的平均计算时间。

关于这些结果，有几个方面需要提到。首先，EA-LGFi 为 500 个问题实例生成的最佳解决方案所使用的箱子数为 7239 个，这是任何算法所达到的最佳值。迄今为止最好的算法（GRASP）得到的值为 7241。值得注意的是，EA-LGFi 能够解决四个以前从未解决过的最优性问题实例。这涉及实例 173 和 174（都来自 II 类，有 60 个项目）、实例 197（来自 IV 类，有 100 个项目）和实例 298（来自 VI 类，有 100 个项目）。

最后，我们想评论一下计算时间。由于使用了不同的计算时间限制来生成结果，计算时间肯定不能直接比较。然而，所有算法的计算时间一般都很低。

对于计算时间的要求，没有一种算法能比其他算法有特定的优势或缺点。在下面，我们提供了关于竞争对手算法的处理器和计算时间限制的信息： SCH 在数字 Alpha 533 MHz 上运行，每个实例的时间限制为 100 秒。抓取是在一个具有 1500 MHz 的奔腾移动设备上执行的，每个应用程序的停止标准为 50000 次迭代。

6.结论与展望

本文提出了一种相当简单的混合进化算法，以解决自由断头台切割（2BP）下的定向二维箱装料问题。该算法强烈地基于文献中现有的单次启发式（LGFi）的概率版本。结果表明，与现有的计算方法相比，该算法获得了很好的效果。事实上，四个以前从未解决到最优性的问题实例，可以解决到最优性。我们可以肯定这一点，因为所找到的最佳解的目标函数值等于最佳下界值。总之，进化算法为 500 个实例生成的最佳解决方案总共使用了 7239 个箱子。

这是为 2BP 提出的任何算法所达到的最佳值。在未来，我们计划研究 LGFi 的概率版本可能采用的其他方法开采例如，蚁群优化方法可能比进化算法更适合使用用于学习 LGFi 的输入序列。此外，我们计划在我们的算法中添加一个局部搜索程序，以改进所构造的解。

致谢

这项工作得到了西班牙政府 TIN2012-37930-C02-02 基金的资助。此外，克里斯蒂安·布卢姆承认巴斯克巴斯克科学基金会伊克巴斯克的支持。

参考文献

[1] A. 洛迪，马蒂洛， D. Vigo，启发式和元启发式方法的一类二维箱包装问题，通知计算杂志 11(4)（1999）345-357。

[2] M. R. Garey，D. S.约翰逊，《计算机与顽固性：np 完整性理论指南，弗里曼，1979。

[3] E. Hopper，B. Turton，一种用于二维工业包装问题的遗传算法，计算机与工业工程 37（1-2）（1999）375-378。

[4] P. E. Sweeney，E. R.帕特诺斯特，切割和包装问题：分类，应用导向的研究书目，运筹学学会杂志 43(7)（1992）691-706。

[5] C. 布鲁姆，普钦格，雷德尔， A. Roli，组合优化中的混合元启发式：一个调查，应用软计算 11(6)（2011）4135-4151。

[6] L. Wong，L. S. Lee，二维箱子包装问题的启发式放置例程，《数学与统计学杂志》5(4)（2009）334-341。

[7] A. 洛迪，马蒂洛， D. 北京，二维箱包装问题的最新进展，离散应用数学 123（1-3）（2002）379-396。

[8] A. 洛迪，马蒂洛， D. 维，二维包装问题：调查，欧洲运筹学杂志 141(2)（2002）241-252。

[9] A. Lodi，《二维箱包装和分配问题的算法》，博士论文，博洛尼亚大学工作室（1999）。

[10] K. A.道斯兰，W。 B. 道斯兰，包装问题，《欧洲运筹学杂志》56(1)（1992）2-14。

[11]ffff.ffff.公司，Jr.，ffff.ffff.加雷，D.ffff.约翰逊，ffff.E.Tarjan，面向水平的二维包装算法的性能边界，

SIAM 计算学杂志 9(4)（1980）808-826。

[12]b.ffff.Baker，ffff.Jr.，ffff.雷维斯特，二维正交包装，模拟计算杂志 9(4)（1980）846-855。

[13] J. B. G. Frenk，G.加兰博斯，二维矩形箱装箱问题的混合下一步算法，计算 39(3)（1987）201-217。

[14] F. R. K. Chung，M. R. Garey，D. S.约翰逊，关于包装二维箱，西门子代数和离散方法杂志 3(1)（1982）66-76。[15] J. O. Berkey，P. Y. Wang，二维有限箱包装算法，运筹学学会杂志 38(5)（1987）423-429。

[16] M. Monaci，P. Toth，一种基于集合覆盖的启发式方法，通知《计算杂志 1》18(1)（2006）71-85.908 克里斯蒂安·布卢姆和维雷纳·施密德/普朗迪亚计算机科学 18（2013）899-908

[17] T. G.克雷尼克，G.珀波利，R. Tadei，三维箱子包装的基于极端点的启发式方法，通知计算杂志 20(3)（2008）368-384。

[18] A. 洛迪，马蒂洛， D. Vigo，面向二维料箱包装问题的近似算法，欧洲运筹学杂志 112(1)（1999)158-166。

[19] O. Faroe，D.皮辛格，M.扎卡里亚森，三维装箱问题的引导局部搜索，《通知计算杂志》15(3)（2003）267-283。

[20] K.-H。刘志，解决二维箱包装问题的重量退火算法，研究/计算机科学界面第47卷，施普林格，纽约，纽约，2009，页。121–146.

[21] M. A.博舍蒂， A. Mingozzi，二维有限箱包装问题。第二部分：新上界，4OR1（2003）135-147。

[22]F.Parreno，R.阿尔瓦雷斯-瓦尔德斯，J. F.奥利维拉，J. M.塔玛里特，两种三维箱包装的混合抓取/VND 算法，运筹学年鉴 179(1)（2010）203-220。

[23] D. Pisinger，M. Sigurd，使用分解技术和约束编程来解决二维箱包装问题，通知计算杂志 19(1)（2007）36-51。

[24] J.普钦格，G. Raidl，三阶段二维填料箱的模型和算法，欧洲运筹学杂志 183(3)（2007）1304-1327。

[25] L. Davis，第 6 章：基于顺序的遗传算法与图形着色问题，范·诺斯特兰德·莱因霍尔德，纽约，1991 年，页。72–90.

[26] S.马泰罗，D.维戈，二维有限箱包装问题的精确解，管理科学 44(3)（1998）388-399。

译文原文

Solving the 2D bin packing problem by means of a hybrid evolutionary algorithm

Christian Blum Verena Schmid

ABSTRACT

Combinatorial optimization problems dealing with 2D bin packing fifind applications, for example, in the context of transportation/warehousing and for the cutting of glass, wood, and metal. In this work we consider the oriented 2D bin packing problem under free guillotine cutting, a problem in which a set of oriented rectangular items is given which must be packed into a minimum number of bins of equal size. Our algorithm proposal to tackle this problem concerns an evolutionary algorithm that makes heavy use of a randomized one-pass heuristic for constructing solutions. The results of the proposed algorithm are compared to some of the best approaches from the literature. This comparison shows that our algorithms is very competitive to state-of-the-art approaches. In particular, the optimal solutions to four previously unsolved instances were found.

Keywords: 2D bin packing, free guillotine cutting, evolutionary algorithm, hybrid computational system

1 INTRODUCTION

In this work we propose a hybrid evolutionary algorithm for solving the 2D bin packing problem in which the items have a fifixed orientation and free guillotine cutting is allowed. According to Lodi et. al [1] this problem is denoted in the literature as 2BP|O|F. Henceforth we will refer to this problem simply by *2BP*. Concerning the complexity, note that the 2BP problem is an NP-hard combinatorial optimization problem (see [2]). In general, bin packing problems have been extensively studied in the past. The main reason for their popularity is a large number of real-world applications. Real world applications for the 2BP include, for example, cutting glass, wood, or metal and packing in the context of transportation or warehousing (see [3, 4]). As mentioned already above, the 2BP is tackled in this work by a hybrid evolutionary algorithm. This algorithm can be attributed to the class of hybrid metaheuristics, a fifield of research which deals with combinations of

metaheuristics, such as evolutionary algorithms and tabu search, with other techniques for optimization, such as heuristics and exact methods. It has been shown that such approaches may strongly benefifit from a synergy between the combined algorithmic approaches. For more information on hybrid metaheuristics we refer the interested reader to [5]. The hybrid aspect of the evolutionary algorithm proposed in this work concerns the representation of solutions. In fact, instead of working directly on solutions to the problem, the search space of the algorithm consists of all permutations of the items belonging to a problem instance. In order to determine the objective function value of a permutation, the LGFi heuristic proposed in [6] is used. This heuristic takes any permutation of all the items as input and produces a solution to the corresponding 2BP problem instance in the specifific way which is outlined in Section 4.1. The experimental results show that the proposed algorithm compares favorably to current state-of-the-art approaches. In fact, summing up the number of used bins concerning all considered 500 problem instances, the evolutionary algorithm reaches a value of 7239, which is the best value reached by any algorithm that has been proposed for this problem. The organization of the paper is as follows. In Section 2 we provide an overview of related work. Next, in Section 3 we provide a technical description of the tackled problem. The proposed algorithm is then presented in Section 4. Finally, an experimental evaluation is provided in Section 5, while conclusions and an outlook to the future are given in Section 6.

2 RELATED WORK

In the following we focus on rather recent work on (meta-)heuristics for the 2BP problem. A good overview on earlier work may be found in [7, 8, 9, 10].

Heuristics. The literature mainly distinguishes between *one-phase approaches* and *two-phase approaches*. Onephase algorithms pack the items directly into bins, whereas two-phase algorithms fifirst pack the items into levels, that is, horizontal packings of items that do not exceed the bin width, and then stack these levels into bins. Well known *level-packing algorithms* are Next-Fit Decreasing Height (NFDH), First-Fit Decreasing Height (FFDH) and Best-Fit Decreasing Height (BFDH) [11]. These strategies were originally developed for the one-dimensional bin packing problem, but have later been adapted to strip packing problems and to the two-dimensional case. All three heuristics require the items to be sorted by non-increasing height, which represents

the order in which they are packed. Moreover, they pack the items into one bin of infifinite height.

Among the most important *one-phase approaches* are Alternate Direction (AD) [1], Bottom-Left Fill (BLF) [12], Improved Lowest Gap Fill (LGFi) [6] and Touching Perimeter (TP) [1]. Concerning *two-phase approaches* we distinguish between Hybrid Next-Fit (HNF) (see [13]) which is based on NFDH, Hybrid FirstFit (HFF) [14] which is based on FFDH and Finite Best-Strip (FBS) [15]—also sometimes referred to as Hybrid Best-Fit—which is based on BFDH. Another example concerns Knapsack Packing (KP) [1]. Finally, Floor Ceiling (FC) [1] can be seen as an improvement over FBS. The best heuristic for the 2BP which is currently available (labelled SCH) is based on solving a set-covering formulation of the problem [16] by means of column generation. In the fifirst phase, a rather small subset of all possible columns is generated by using greedy procedures and fast constructive heuristic algorithms from the literature. In the second phase, the resulting set-covering instance is solved by means of a Lagrangian-based heuristic. In addition, some heuristics developed for three-dimensional packing can sometimes easily be applied to the 2BP. An example is the extreme point based heuristic from [17]. This heuristic uses extreme points to determine all points in the bin where items can be placed. Extreme points can either be corners of the already placed items or points generated by the extended edges of the placed items. These points are updated every time an item is placed into the bin. For placing the items a modifified version of BFDH is used.

Metaheuristics. Apart from heuristics, more recently researchers also explored the application of metaheuristics to the 2BP problem. The earliest metaheuristic developed for the 2BP was *tabu search* (TS) [18, 7]. Moreover, diffffferent researchers have applied *guided local search* (GLS) [19] and *weight annealing* (WA) [20]. A rather simple metaheuristic, labeled HBP, based on a greedy heuristic has been proposed in [21]. HBP assigns a scoreto each item. Then, for the construction of a solution, the items are considered according to non-increasing values of the scores. After the construction of a solution the scores are updated using a certain criterion. This procedure is iterated until a pre-defifined stopping criterion is met. Finally, the currently best-performing metaheuristic is a hybrid between a greedy randomized adaptive search procedure (GRASP) and variable neighborhood descent (VND) [22]. The solution

construction phase of GRASP is hereby based on a maximal-space heuristic from the fifield of container loading.

3 THE 2D BIN PACKING PROBLEM

A technical description of the considered problem is given in the following by means of an ILP model

from [23].1 In this model, let I = {1,..., *n*} be the set of items that have to be packed into a minimum number of equal-sized bins of width *W* and height *H*. Each item *i* ∈ I is characterized by its width *wi* and height *hi*. In the context of this paper note that *W* and *H* as well as *wi* and *hi* (for all *i* ∈ I) are integer values, which is suffifffifficient for solving the considered problem instances. The ILP model consists of the following variables. For each *i* ∈ I

1. the value of variable *mi* indicates the number of the bin in which item *i* is placed;

2. the values of variables *xi* and *yi* indicate the lower left coordinates of the position of item *i* in bin *mi*.

Moreover, for each pair *i, j* ∈ I (such that *i* *j*)

1. the binary variable *li j* is set to 1 iffff item *i* is located *left* of item *j*;

2. the binary variable *bi j* is set to 1 iffff item *i* is located *below* item *j*;

3. the binary variable *pi j* assumes value 1 iffff *mi* < *mj*.

Finally, variable *v* indicates the number of bins in use. Using these variables the following ILP solves the 2D bin packing problem without item rotation and under free guillotine cutting.

Hereby, constraints (2), (3), (4), (6) and (7) take care that no overlaps occur between two items and/or between items and the borders of the bins. Constraints (5) are responsible for a correct setting of the *pi j*-variables. Furthermore, constraints (8) and (9) limit the bin numbers to [1, *v*]. Finally, constraints (10) are symmetry breaking constraints that facilitate the work of the MIP solver.

4 THE PROPOSED ALGORITHM

As mentioned already in the introduction, the proposed evolutionary algorithm is strongly based on heuristic

LGFi, which was described by Wong and Lee in [6]. In the following we fifirst outline the way in which the LGFi

heuristic works. Afterwards, the evolutionary algorithm is described in detail.

1Note that an alternative model was presented in [24].902

Christian Blum and Verena Schmid / Procedia Computer Science 18 (2013) 899 – 908

Algorithm 1 Evolutionary Algorithm for the 2BP (EA-LGFi)

1: **input:** values for parameters *p*size, *c*rate, κ and δ

2: *P* := GenerateInitialPopulation(*p*size, κ)

3: **while** stopping criterion not met **do**

4:

P := Crossover(*P*, *c*rate, δ)

5: *P* := AddNewSolutions(*P* , *p*size, κ)

6: **end while**

7: **output:** best solution found

4.1. Heuristic LGFi

Note that LGFi is a two-stage heuristic. In the *preprocessing stage* items are sorted into a list, while in

the *packing stage* these items are packed from the list into bins. More specifically, in the preprocessing stage items are sorted by non-increasing area as a first criterion. Ties are broken by non-increasing absolute difference between height and width of the items. The packing stage is an iterative process in which the following actions

are performed at each iteration. First, the bottom leftmost position at which an item may be placed is identified. This position is henceforth called the *current position*. Then, two gaps are calculated with respect to this position. The *horizontal gap* is defined as the distance between the current position and either the right border of the bin or the left edge of the first item between the current position and the right border of the bin. The distance between

the current position and the upper border of the bin defines the value of the *vertical gap*. The value of the smaller gap is called *current gap*. The current gap is compared to either the widths of the items from the list of unpacked items, if the horizontal gap is the current gap, or to the heights of the items from the list of unpacked items, if the vertical gap is the current gap. The first item that fills the gap completely is placed with its bottom left corner at the identified position. If no such item exists, the first item which fits without any overlap is placed with its bottom

left corner at the current position. If no such item exists either, some of the area must be declared *wastage area*, which works as follows. A wastage area with the width of the horizontal gap is created. The height of the wastage area is chosen as the height of the upper edge of the lowest neighboring item, or, if no neighboring item exists, as the height of the bin. Finally, if no current position can be found, and if unpacked items exist, a new bin is opened.

4.2. The Evolutionary Algorithm

A solution in the context of the proposed algorithm—henceforth labelled EA-LGFi—is an input sequence *s* for LGFi. Note that any input sequence *s* is a permutation of all items that must be packed. The item at position *j* of this permutation (where $j = 1,..., n$) is denoted by *sj*. The objective function value $f(s)$ of a solution *s* is calculated by applying LGFi to *s*. The pseudo-code of EA-LGFi is shown in Alg. 1. The fifirst step of EALGFi consists in generating the initial population of size *p*size (see function GenerateInitialPopulation(*p*size, κ)).

Then, at each iteration a crossover operator is applied in function Crossover(*P*, *c*rate, δ), generating $c\text{rate} \cdot p\text{size}$ new solutions, where $0 < c\text{rate} \leq 1$. This results in a new population *P* of less than *p*size solutions. The missing $p\text{size} - |P|$ solutions are generated by function AddNewSolutions(P , *p*size, κ). In the following the three functions of algorithm EA-LGFi are outlined in more detail.

GenerateInitialPopulation(*psize*, κ). In this function, *p*size solutions are probabilistically generated as follows. First, let us denote the input sequence in which items are ordered with respect to non-increasing area as the *deterministic input sequence*. The position of an item *i* in the deterministic input sequence is henceforth called *posi*. Any solution *s* for the initial population is probabilistically generated based on the deterministic input

sequence. This is done as follows. First, remember that the total number of items is denoted by *n*. A value *vi* is then assigned to each item *i* in the following way:

$$vi := (n - posi)\ \kappa \tag{11}$$

where $\kappa \geq 1$ is one of the input parameters of the function. The positions of *s* are fifilled from 1 to

n in an iterative way. At each step, let $I \subseteq I$ be the set of items that are not yet assigned to *s*

. An item $i \in$ I is chosen according903*Christian Blum and Verena Schmid / Procedia Computer Science 18 (2013) 899 – 908* to probabilities **p**(*i* proportional to *vi*: |I) (for all $i \in$ I) by roulette-wheel-selection. These probabilities **p**(*i*|I) are calculated

p(*i*|I) = *vi* $i \in$ I *vi* (12)

Note that the larger the value of parameter κ, the more similar the newly generated input sequence *s* will be to the deterministic input sequence.

Crossover(*P*, *crate*, δ). This operator, which is inspired by *uniform order-based crossover* [25], applies recombination to each of the best *c*rate · |*P*| solutions of *P*, where 0

< *c*rate ≤ 1 is a parameter of the algorithm. Extensive

empirical tests have shown that a crossover rate

*c*rate = 0.7 works best for the instances at hand. For each solution *s* from the set of best *c*rate · |*P*| solutions of *P*, a crossover partner *sc* ∈ *P* (such that *sc* *s*) is chosen from *P*

by means of roulette-wheel-selection. Assume that

P is an ordered list in which solutions are sorted according

to their objective function values in a non-increasing manner. Ties are broken by the load of the last bin, that is, solutions with a lower load in the last bin are ordered fifirst. Let *pos*(*s*) denote the position of a solution *s* in *P*. where δ ≥ 1 is a parameter of the algorithm. Given two crossover partners

s and *sc* , one offffspring solution *s*offff is

generated as explained in the following. First, three pointers (

k, *l* and *r*) are initialized to the fifirst position. Then,

the *n* positions of *s*offff are fifilled from 1 to *n* as follows. If *sk* = *sc*

l then *s*offff *r* := *sk*. In other words, if position *k* of solution *s* and position *l* of solution *sc* contain the same item, then this item is placed at position *r* of the

offffspring solution *s*offff. Next, position pointer *r* is incremented, and position pointers *k* and *l* are moved to the right until reaching the closest position containing an item which does not yet appear in solution *s*offff. In case *sk* *sc l* , the item for position *r* of solution *s*offff is chosen probabilistically among *sk* and *sc*

l , where a probability of 0.75 is given to the item originating from the better of the two solutions. Afterwards, the position pointer *r* is incremented. Moreover, the position pointer of the solution from which the item was selected is moved to the right until

reaching the closest position containing an item which does not yet appear in solution *s*offfff. The resulting solution *s*offfff is evaluated by using it as input for LGFi. In case *f*(*s*offfff) < *f*(*s*) or *f*(*s*offfff) = *f*(*s*) and *s*offfff has a lower load than *s* in the last bin, solution *s*offfff is added to the new population *P* , otherwise solution *s* is added to *P* .

AddNewSolutions(*P* , *psize*, κ). This function probabilistically generates *p*size − |*P* | solutions in the same way as in function GenerateInitialPopulation(*p*size, κ).

5. Experimental Evaluation

EA-LGFi was implemented in ANSI C++ using GCC 4.4 for compiling the software. The experimental

results that we outline in the following were obtained on a PC with an AMD64X2 4400 processor and 4 Gigabyte of memory. The proposed algorithm was applied to a benchmark set of 500 problem instances from the literature.2

After an initial study of the algorithms' behavior, a detailed experimental evaluation is presented.

5.1. Problem Instances Ten classes of problem instances for the 2BP are provided in the literature. A fifirst instance set, containing six classes (I-VI), was proposed by Berkey and Wang in [15]. For each of these classes, the widths and heights of the items were chosen uniformly at random from the intervals presented in Table 1. Moreover, the classes diffffer in the width (*W*) and the height (*H*) of the bins. Instance sizes, in terms of the number of items, are taken from {20, 40, 60, 80, 100}. Berkey and Wang provided 10 instances for each combination of a class with an instance size. This results in a total of 300 problem instances.

2These 500 instances can be downloaded from http://www.or.deis.unibo.it/research.html.904

Christian Blum and Verena Schmid / Procedia Computer Science 18 (2013) 899 – 908

Table 1. Specifification of instance classes I-VI (as provided by [15]).

The second instance set, consisting of classes VII-X, was introduced by Martello and Vigo in [26]. In general, they considered four diffffferent types of items, as presented in Table 2. The four item types diffffer in the limits for the width *wi* and the height *hi* of an item. Then, based on these four item types, Martello and Vigo introduced four classes of instances which diffffer in the percentage of items they contain from each type. As an example, let us consider an instance of class VII. 70% of the items of

such an instance are of type 1, 10% of the items are of type 2, further 10% of the items are of type 3, and the remaining 10% of the items are of type 4. These percentages are given per class in Table 3. As in the case of the fifirst instance set, instance sizes are taken from {20, 40, 60, 80, 100}.

The instance set by Martello and Vigo consists of 10 instances for each combination of a class with an instance size. This results in a total of 200 problem instances.

Table 2. Item types for classes VII-X (as introduced in [26]). Note that for all classes it holds that $W = H = 100$.

Table 3. Specifification of instance classes VII-X (as provided by [26]).

Class Type 1 Type 2 Type 3 Type 4

VII 70% 10% 10% 10%

VIII 10% 70% 10% 10%

IX 10% 10% 70% 10%

X 10% 10% 10% 70%

5.2. Tuning Experiments

First of all we chose a fifixed number of solution evaluations as stopping criterion for each run of EA-LGFi. In particular, after initial experiments we chose a budget of 106 solution evaluations for each run of EA-LGFi.

Moreover, the value of parameter κ was set to 10 after initial experiments by hand.

The parameters that were selected for tuning are the population size *p*size and δ. Remember that the value of δ is used for the calculation of the probabilities for solutions to be selected as crossover partners. More specififically, we considered *p*size ∈ {10, 100} and δ ∈ {1, 5, 10, 15, 20}. In general, the higher the value of δ, the more are good solutions preferred over worse ones, when selecting a crossover partner *sc* for a solution *s*. Then, EA-LGFi was applied exactly once for each combination of δ and *p*size, to each of the 500 problem instances. The sum of the number of bins used in the best solutions generated for all 500 instances is shown in Figure 1 for each parameter value combination. Even though diffffferences in algorithm performance are quite small, higher values of δ seem to work better than smaller ones. Moreover, a population size of 10 generally seems to work slightly better than a905

population size of 100. The fifinal results of EA-LGFi presented in the following section are the ones obtained with

$\delta = 20$ and $psize = 10$.

5.3. Numerical Results

The results of EA-LGFi are compared in the following with the two best algorithms currently available for the 2BP problem: The set covering heuristic (SCH) from [16] and the hybrid GRASP approach from [22].

Numercial results are shown in Table 4 in a way which is traditional for the 2BP problem. Each table row presents the results for the 10 instances of a combination between instance class (I – X) and number of items (20 – 100). Concerning the columns, the one labelled **LB** contains the sum of the best lower bound values known from the literature for the 10 corresponding problem instances. The results of the compared algorithms are provided in two columns per algorithm. The fifirst column (with heading **value**) provides the sum of the number of bins used in the best solutions generated for the 10 corresponding problem instances. For example, the best solutions generated for the 10 instances of Class I (20 items) by algorithm SCH occupy in total 71 bins. In case a value corresponds to the best performance obtained by any algorithm, it is highlighted in bold. Moreover, in the case of EA-LGFi a value is marked by an asterisk if it is better than the best know value as of today. The second column (with heading **time (s)**) shows the average computation time (in seconds) necessary to fifind the best solutions for the 10 problem instances of a combination between instance class and number of items. For example, algorithm SCH needed on average 0.06 seconds to fifind its best solutions for the 10 instances of Class I (20 items). Finally, the last line of Table 4 provides a summary of the results over all 500 problem instances. For each algorithm is given the sum of number of bins used, as well as the average computation time for the 500 instances.

There are several aspects about the results that should be mentioned. First, the number of bins used by the best solutions generated by EA-LGFi for the 500 problem instances amounts to 7239, which is the best value ever achieved by any algorithm. The best algorithm so far (GRASP) achieved a value of 7241. It is also interesting to note that EA-LGFi is able to solve four problem instances to optimality that have never been solved before. This concerns instances 173 and 174 (both from Class IV, with 60 items), instance 197 (from Class IV, with 100 items), and instance 298 (from Class VI, with 100 items).

Finally, we would like to comment on the computation times. Due to the fact that diffffferent processors and different computation time limits have been used for the generation of the results, the computation times are certainly not directly comparable. However, the computation times of all algorithms are, in general, very low.

no algorithm can be identifified to have a particular advantage or disadvantage over the other algorithms for what concerns the computation time requirements. In the following we provide the information about processors and computation time limits for the competitor algorithms: SCH was run on a Digital Alpha 533 MHz with a time limit of 100 seconds per instance. GRASP was executed on a Pentium Mobile with 1500 MHz with a stopping criterion of 50000 iterations per application.

6. Conclusions and Outlook

In this paper we have presented a rather simple hybrid evolutionary algorithm for tackling the oriented two dimensional bin packing problem under free guillotine cutting (2BP). The proposed algorithm is strongly based on a probabilistic version of an existing one-pass heuristic (LGFi) from the literature. The results have shown that the proposed algorithm obtains very good results in comparison to current state-of-the-art approaches. In fact, four problem instances that were never solved to optimality before, could be solved to optimality. We can be sure of that because the objective function value of the best solution found is equal to the best lower bound value. In summary, the best solutions generated by the evolutionary algorithm for the 500 instances use 7239 bins in total.

This is the best value ever achieved by any algorithm proposed for the 2BP.

In the future we plan to investigate additional ways in which the probabilistic version of LGFi might be

exploited. For example, an ant colony optimization approach might be better suited than an evolutionary algorithm

for learning input sequences for LGFi. Moreover, we plan to add a local search procedure to our algorithms for improving the constructed solutions.

Acknowledgements

This work was supported by grant TIN2012-37930-C02-02 of the Spanish Government. In addition, Christian Blum acknowledges support from IKERBASQUE, the Basque Foundation for Science.

References

[1] A. Lodi, S. Martello, D. Vigo, Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, INFORMS Journal on Computing 11 (4) (1999) 345–357.

[2] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[3] E. Hopper, B. Turton, A genetic algorithm for a 2D industrial packing problem, Computers and Industrial Engineering 37 (1–2) (1999) 375–378.

[4] P. E. Sweeney, E. R. Paternoster, Cutting and packing problems: A categorized, application-orientated research bibliography, The Journal of the Operational Research Society 43 (7) (1992) 691–706.

[5] C. Blum, J. Puchinger, G. Raidl, A. Roli, Hybrid metaheuristics in combinatorial optimization: A survey, Applied Soft Computing 11 (6) (2011) 4135–4151.

[6] L. Wong, L. S. Lee, Heuristic placement routines for two-dimensional bin packing problem, Journal of Mathematics and Statistics 5 (4) (2009) 334–341.

[7] A. Lodi, S. Martello, D. Vigo, Recent advances on two-dimensional bin packing problems, Discrete Applied Mathematics 123 (1-3) (2002) 379–396.

[8] A. Lodi, S. Martello, D. Vigo, Two-dimensional packing problems: A survey, European Journal of Operational Research 141 (2) (2002) 241–252.

[9] A. Lodi, Algorithms for two-dimensional bin packing and assignment problems, Ph.D. thesis, Universita degli Studio di Bologna (1999).

[10] K. A. Dowsland, W. B. Dowsland, Packing problems, European Journal of Operational Research 56 (1) (1992) 2–14.

[11] E. G. Coffffman, Jr., M. R. Garey, D. S. Johnson, R. E. Tarjan, Performance bounds for level-oriented two-dimensional packing algorithms,

SIAM Journal on Computing 9 (4) (1980) 808–826.

[12] B. S. Baker, E. G. Coffffman Jr., R. L. Rivest, Orthogonal packings in two dimensions, SIAM Journal on Computing 9 (4) (1980) 846–855.

[13] J. B. G. Frenk, G. Galambos, Hybrid next-fifit algorithm for the two-dimensional rectangle bin-packing problem, Computing 39 (3) (1987) 201–217.

[14] F. R. K. Chung, M. R. Garey, D. S. Johnson, On packing two-dimensional bins, SIAM Journal on Algebraic and Discrete Methods 3 (1) (1982) 66–76. [15] J. O.

Berkey, P. Y. Wang, Two dimensional fifinite bin packing algorithms, Journal of the Operational Research Society 38 (5) (1987) 423–429.

[16] M. Monaci, P. Toth, A set-covering-based heuristic approach for bin-packing problems, Informs Journal on Computing 18 (1) (2006) 71–85.908 *Christian Blum and Verena Schmid / Procedia Computer Science 18 (2013) 899 – 908*

[17] T. G. Crainic, G. Perboli, R. Tadei, Extreme point-based heuristics for three-dimensional bin packing, Informs Journal on Computing 20 (3) (2008) 368–384.

[18] A. Lodi, S. Martello, D. Vigo, Approximation algorithms for the oriented two-dimensional bin packing problem, European Journal of Operational Research 112 (1) (1999) 158–166.

[19] O. Faroe, D. Pisinger, M. Zachariasen, Guided local search for the three-dimensional bin-packing problem, Informs Journal on Computing 15 (3) (2003) 267–283.

[20] K.-H. Loh, B. Golden, E. Wasil, A Weight Annealing Algorithm for Solving Two-dimensional Bin Packing Problems, Vol. 47 of Research/Computer Science Interfaces, Springer, New York, NY, 2009, pp. 121–146.

[21] M. A. Boschetti, A. Mingozzi, The two-dimensional fifinite bin packing problem. part II: New lower and upper bounds, 4OR 1 (2003) 135–147.

[22] F. Parre˜no, R. Alvarez-Valdes, J. F. Oliveira, J. M. Tamarit, A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing, Annals of Operations Research 179 (1) (2010) 203–220.

[23] D. Pisinger, M. Sigurd, Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem, INFORMS Journal on Computing 19 (1) (2007) 36–51.

[24] J. Puchinger, G. Raidl, Models and algorithms for three-stage two-dimensional bin packing, European Journal of Operational Research 183 (3) (2007) 1304–1327.

[25] L. Davis, Chapter 6: Order-Based Genetic Algorithms and the Graph Coloring Problem, Van Nostrand Reinhold, New York, NY, 1991, pp. 72–90.

[26] S. Martello, D. Vigo, Exact solution of the two-dimensional fifinite bin packing problem, Management Science 44 (3) (1998) 388–399.