山东大学
SHANDONG UNIVERSITY

毕业论文(设计)

论文（设计）题目：

面向多段线路径的直线段和圆弧 G1 连续拟合

姓　　名 乔 婧

学　　号 201900130111

学　　院 计算机科学与技术学院

专　　业 人工智能

年　　级 2019 级

指导教师 赵海森

2023 年 5 月 9 日

摘　　要

随着计算机技术和制造技术的飞速发展，多段线路径在数控加工、机器人、计算机图形学等领域中得到了广泛的应用。在数控加工领域，输入通常是散点路径，研究人员通常使用曲线拟合方法对散点路径进行拟合规划，但是这些方法的计算量较大，运行的时间成本和空间成本高，不利于数控机床的存储以及实时处理加工。

本研究旨在针对面向多段线路径的直线段和圆弧 G1 连续拟合问题，探究一种新的算法。该方法首先将由多个小离散段表示的路径用直线段进行拟合，然后在拟合出的直线段基础上进行圆弧段拟合，根据每个连接点处的切线方向和曲率来确定各个段的形状和位置，以确保拟合曲线的 G1 连续平滑度。

本研究相较于曲线拟合方法，提出了面向多线段路径的直线段和圆弧 G1 连续拟合的方法。该算法能够在保证拟合精度的同时降低计算复杂度，从而提高算法的实时性，并能够处理复杂的路径。为了评估该方法的有效性，本工作进行了多组实验，实验结果表明，本文提出的方法可以获得与预期相符的 G1 连续性和模型处理时间。本研究的成果将对计算机辅助制造的发展和提升产生积极的影响。

关键词：曲线分割；直线检测；圆弧检测；曲线拟合；平滑连接

ABSTRACT

With the rapid development of computer technology and manufacturing technology, polyline paths have been widely used in fields such as numerical control machining, robotics, and computer graphics. In the field of numerical control machining, the input is usually a scattered path, and researchers typically use curve fitting methods to plan the scattered path. However, these methods have a large computational cost, high time and space costs, and are not conducive to the storage and real-time processing of numerical control machines.

This study aims to explore a new algorithm for the G1 continuous fitting problem for multi-segment paths based on straight lines and arcs. The proposed method first fits the path represented by multiple small discrete segments with straight lines, and then fits the arcs based on the fitted straight lines, determining the shape and position of each segment based on the tangent direction and curvature at each connection point to ensure the G1 continuity smoothness of the fitted curve.

Compared with curve fitting methods, this study proposes a G1 continuous fitting method based on straight lines and arcs for multi-segment paths. The algorithm can reduce the computational complexity while ensuring the fitting accuracy, thereby improving the real-time performance of the algorithm and being able to process complex paths. To evaluate the effectiveness of this method, multiple experiments were conducted, and the results showed that the proposed method can achieve the expected G1 continuity and model processing time. The results of this study will have a positive impact on the development and improvement of computer-aided manufacturing.

Keywords: Curve Segmentation, Line Detection, Arc detection, Curve Fitting, Smooth Connection.

目　录

第1章绪　论

1.1 选题背景和研究意义

当今社会，随着计算机技术和制造技术的飞速发展，多段线路径在数控加工、机器人、计算机图形学等领域中得到了广泛的应用[1]。如图 1-1 中左图输入数据所示，多段线路径是由多个离散小线段构成的，多段线可以将多段连续的小线段作为一个整体进行绘制和编辑。在数控加工领域，多段线拟合可用于平滑的刀具路径生成和数据压缩[2]，可以减少机床在加工曲线路径时的振动和冲击，从而可以降低机床的磨损和损坏，延长机床的使用寿命；在机器人领域，多段线拟合路径可以用于机器人轨迹插值，将复杂的轨迹用多个线段进行拟合，通过插值算法将线段拼接起来，生成机器人运动轨迹，从而提高机器人的轨迹生成效率和精度[3]；在三维打印领域，多段线拟合可以用于构建复杂的三维模型[4]，且能使打印路径更加平稳，减少打印头在曲线处的震动和晃动，从而提高打印稳定性。

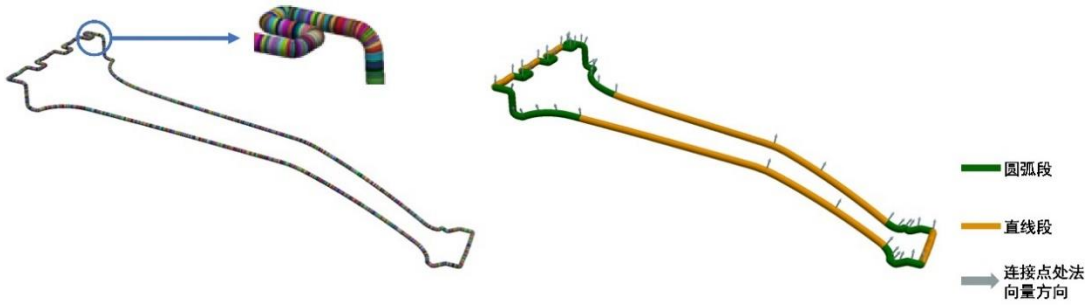


图 1-1 用直线段和圆弧拟合多线段，并做到 G1 连续。左图为输入的多段线数据

本研究主要围绕在数控加工制造过程中，如何高效率地拟合多段线路径这一问题进行展开。在数控加工领域，输入通常是多段线路径，而以多段线路径作为输入存在两方面问题。首先，多段线要求的存储空间大，且由于离散的小线段数量过多，不容易进行路径规划和轨迹跟踪；其次，数控加工中对路径规划的主要制造需求是在满足约束条件下，控制路径加工的指令条数越少越好，而离散的线段越多，所要计算的法向量也越多，指令条数也随之增多。因此要对多段线路径进行拟合规划，从而可以更好地控制路径的精度和平滑度，使得路径更加符合制造要求。研究人员通常使用贝塞尔曲线和 B 样条等曲线拟合方

法对多段线路径进行拟合规划[5]，但是对只使用通用指令集进行路径处理的机床设备来说，这些方法的计算量较大，运行的时间成本和空间成本高，不利于数控机床的存储以及实时处理加工。相对于曲线拟合，用直线段和圆弧段拟合多段线路径具有简单易实现、可控制的精度和平滑度、可避免过度拟合、可方便地进行后续处理和加工效率高等优势，在实际应用中具有广泛的应用价值。

G1 连续在计算机辅助制造和数控加工领域中非常重要，是提高加工精度、加工效率、加工表面的光滑度和支持多种加工方式必须要考虑的因素之一。G1 连续是指路径上相邻两个路径段之间的切线方向连续，如果路径上相邻两个路径段之间的切线方向不连续，则会产生明显的角度变化，从而影响加工质量。将直线段和圆弧进行 G1 连续拟合可以实现更加平滑的路径、提高加工精度、避免机床振动和避免路径交叉。

从理论意义上说，直线段和圆弧是最基础的线段类型，因此研究多段线路径的直线段和圆弧 G1 连续拟合问题，可以为其他曲线和曲面拟合问题提供参考和借鉴。此外，该问题的研究也可以深入探究曲线和曲面的 G1 连续性，从而进一步完善相关理论体系。从实际应用价值上说，将多段线路径的直线段和圆弧进行 G1 连续拟合具有简单易实现、可控制的精度和平滑度、可避免过度拟合、可方便地进行后续处理和加工效率高等优势。因此，如何采用简单的直线段和圆弧段进行高精度、高效率的曲线分割[6]以及 G1 连续的路径拟合加工，成为了当前国内数控加工领域亟待解决的问题。

本研究旨在针对面向多段线路径的直线段和圆弧 G1 连续拟合问题，探究一种新的算法，如图 1-1 所示，输入多段线路径数据，算法处理后输出用直线段和圆弧段进行拟合后的 G1 连续路径，该算法能够在保证拟合精度的同时降低计算复杂度（相比曲线拟合），从而提高算法的实时性，并能够处理复杂的路径。该算法的研究成果将为数控机床的加工提供一种高效率的加工路径规划方案。通过对直线段和圆弧段的合理组合，我们可以实现更加高效的路径拟合和加工，从而提高数字化制造的效率和精度，具有重要的实际应用价值。

综上所述，本研究的选题背景和研究意义在于解决当前数控机床无法应用高级曲线拟合技术进行高效率加工的问题，并针对此问题提出了面向多段线路径的直线段和圆弧 G1 连续拟合的方法。

1.2 文献综述

当代计算机图形学技术越来越多地应用于三维建模和路径规划领域。而在这些应用中，对多段线路径进行拟合是一项非常重要的任务。因为原始数据是多个连接在一起的离散小线段，且相比用贝塞尔曲线或 B 样条进行曲线拟合，用简单元素——直线段和圆弧段进行拟合可以在通用机床的加工中更好的提高加工效率，因此路径拟合的任务可以看作是将多个离散小线段转化为一条连续的曲线，该曲线由直线段和圆弧段组成，并尽可能贴近原始数据。在 G1 连续拟合多段线[7]中，目标是不仅保持连续性，还要保证曲线的一阶导数连续，即曲线没有明显的转角或方向突变。

所需的描述类型可能取决于应用，但通常需要基于直线近似和高阶曲线（如圆弧、样条曲线等）的组合来描述曲线[5]。已经对算法的开发进行了大量研究，这些算法可以产生紧凑、准确、有意义且适合进一步处理的表示。早期的工作集中在通过在最大偏差点[8]和不连续点[9]处分割曲线来生成直线近似。最近，已经提出了一种用于分割二维和三维曲线的算法[10]。该算法是对其他算法的改进，因为它使用圆弧和直线来描述在使用曲率和曲率变化率的最大值检测到的不连续处分割这些不连续处。

直线段拟合是路径拟合中最简单的情况。在参考文献中，研究者们采用了不同的方法来实现直线段拟合。其中一种方法是最小二乘法拟合[11]，该方法通过最小化原始数据路径到直线段的距离平方和来确定最佳拟合直线。另一种方法是迭代重心法，该方法通过迭代计算路径上点的重心和路径上点到重心的偏差向量之间的关系来拟合最佳直线段[6]。

对于圆弧拟合，参考文献中提出了许多不同的方法。其中一种方法是最小二乘法拟合[11]，该方法通过最小化路径上的点到圆弧的距离平方和来确定最佳拟合圆弧[6]。另一种方法是基于样条曲线的拟合方法，该方法通过使用样条曲线来拟合路径中的弧段[12]，从而实现拟合圆弧的目标。

West[13]和 Rosin[6]描述了一种递归算法用于将图像中的曲线分割成圆弧和直线组合的方法。该算法首先分析连接边点的列表并找到多边形描述，然后分析将圆弧拟合到连接线组的描述。结果是由圆弧和直线组成的图像边缘的描述。该算法不使用阈值，在每个决策阶段都会选择最佳选项。

在文献[5]中，李浩和吴文江提出了一种使用贝塞尔曲线进行刀具轨迹拟合的算法。该算法对于连续微小线段加工区域,首先,对离散的指令点进行局部插值,将折线加工路径转化为G1连续的二次有理贝塞尔曲线;然后,调整相邻两条二次有理贝塞尔曲线的权值和连接点处的切线方向,使加工路径达到G2连续性;最后,通过建立公差带将不满足精度要求的二次有理贝塞尔曲线进行重构。

Wan 和 Ventura 提出了一种将平面曲线分割为直线段和椭圆弧的方法[14]。该方法通过先将自适应平滑应用于沿曲线的切线方向，然后在导数上定位高尖峰，来检测拐角。平滑连接则首先通过动态聚焦拟合技术粗略定位，然后通过调整算法进行细化。

除此之外，还有一些研究者探索了利用优化技术、遗传算法和其他基于机器学习的方法来实现三维散点路径的拟合。例如，利用神经网络来拟合圆弧，或者使用深度学习技术来学习路径的特征[15]，从而实现更加精确的路径拟合。

综上所述，对多段线路径进行曲线或曲面拟合的技术和方法非常丰富。在实际应用中，路径拟合的精度、计算成本和平滑度等都是需要考虑的因素。因此，在选择路径拟合方法时，需要根据具体的应用场景和限制条件来选择最适合的方法。在未来的研究中，可以进一步探索深度学习和人工智能等技术在路径拟合中的应用，以提高拟合的精度和效率。

1.3 论文组织结构

文章主要分为五章，分别从选题背景、研究意义、文献综述、算法内容和实验结果与结论几个方面进行阐述。在本节中将概述每一章节的主要内容。

本文的章节结构安排如下：

第一章是绪论，主要包括直线段和圆弧拟合算法的研究背景、意义、实际应用价值等。同时也简要概括了在本文中我们的主要学术贡献以及创新。

第二章主要介绍基元检测算法。我们将其分为直线段检测和圆弧检测，分别介绍了两个检测算法检测的基本原理。其中，针对两种类型的基元检测算法，详细介绍了最具代表性的圆弧段检测算法。

第三章主要介绍了如何使拟合后的由直线段和圆弧段组成的曲线达到G1连续，实现更为平滑的连接。

第四章主要介绍了面向三维散点路径的直线段和圆弧 G1 拟合算法的评价及算法性能评测。在该章节中，我们将使用直线段和圆弧进行路径拟合同使用样条曲线进行路径拟合做了大量的对比实验以说明我们提出的算法的优越性。

第五章是总结与展望，总结本文在使用直线段和圆弧进行路径的 G1 连续拟合在机床加工领域所做的工作，并且探讨以后三维散点路径拟合算法可能发展的方向与趋势。

第2章 基元检测算法

2.1 概述

本文提出的基元检测方法是在Lowe提出的曲线分析和直线描述方法[16]的基础上进行的扩展。Lowe[16]的算法使用递归方法对曲线进行分割，并计算分割后曲线与近似直线之间的最大偏差。该算法需要在每个层次上决定使用单个直线描述还是由两条或更多条直线组合的描述更好。算法使用“显著性”这一指标来衡量直线描述的好坏，即直线最大偏差与直线长度的比值。

本文提出的方法在 Lowe[16]的基础上，将曲线分割为圆弧和直线的组合。该方法同样采用递归算法，首先通过连接边点列表找到多边形描述，然后对连接线组拟合圆弧。最终得到由圆弧和直线组成的图像边缘描述。与 Lowe[16]的算法不同，本文提出的方法不使用阈值，而是在每个决策阶段选择最佳选项。

本文提出的方法相较于 Lowe[16]的算法，能够更加准确、高效地检测到图像中的基元。通过将圆弧和直线组合使用，该方法能够更好地描述复杂的曲线和边缘。算法的递归性质使得曲线的分割更加精确，降低了基元漏检和误检的风险。总的来说，本文提出的方法是 Lowe[16]算法的重要改进，可以应用于多种图像处理和计算机视觉任务中。

2.2 直线段检测

直线段检测可以被视为寻找由多段线组成的曲线的多边形近似值。在Lowe[16]的工作中，假设边缘曲线上的点列表是由连接起点和终点的直线段组成的。该列表在最大偏差点处被分割成两个列表，并且在这两个列表中的每个列表上递归地重复该过程。当线段长度少于四个点或要计算偏差的点数量少于三个点时，递归过程停止。线段长度为三个点的线是可以包含非零偏差的最小直线段。主要算法伪代码实现如算法 2-1 所示。

算法 2-1　直线段检测算法

Line Detection Algorithm

INPUT：*PolylineList*: 用点表示的多段线路径列表，每相邻两个点连接成一个小线段;

OUTPUT：*fitting_Segments*: 存储用直线段拟合多段线路径的拟合结果

```
1:    for each point p in PolylineList do
2:       d= distance (points[i].p, start, end)：计算点到连接起点和终点的直线段的距离
3:       max_deviation=max(d,max_deviation)：找到最大偏差
4:    if max_deviation>固定偏差阈值  then
5:       在最大偏差点处分割成两部分，分别重复上述算法步骤
6:    else if end-start<4 个点  then
7:             return fitting_Segments
8:    else
9:             return fitting_Segments
```

该递归过程的结果是一个多级树，其中每个级别的散点列表的描述是上一级别的更精细的多边形近似。随着递归的展开，树被遍历回到根。在每个级别，如果从较低级别传递上来的线段中的任何一个线段比当前级别的线段更重要，则它们将被保留并作为候选线段传递到下一个较高级别。如果不是这种情况，则会向上传递当前级别的线段。一旦递归完全展开，就可以获得空间曲线的多边形近似。图 2-1 显示了三个递归级别的线性拟合示例。

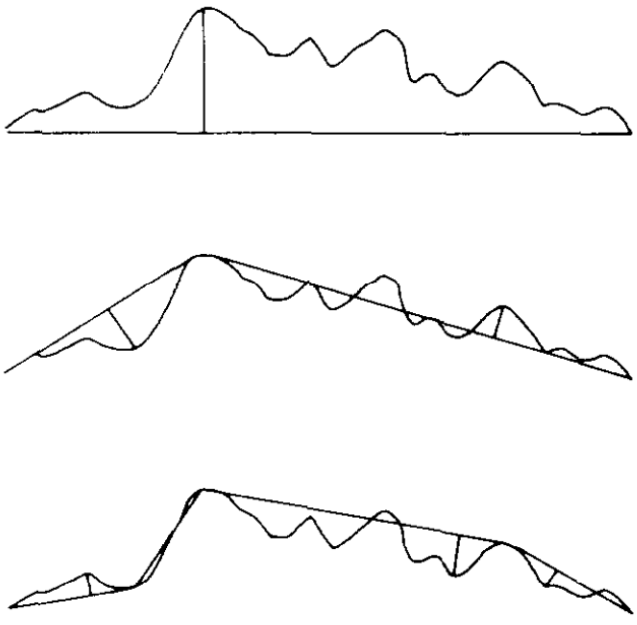


图 2-1 直线段拟合的三个递归过程

Lowe[16]提出的显著性度量是由线段长度除以空间曲线与直线段的最大偏差的比值确定的。然而，当所有点都直接位于直线上时（这种情况经常出现在较短线段上），得到的偏差为 0，显著性度量的计算无意义。为了避免被零除错误，

显著性度量被重新定义为最大偏差除以线段长度的比值，即 Lowe[16]显著性的倒数。因此，显著性值越低，这条线就越显著。该程序的权重有利于长线段（即重要线段）。线段长度越长，可以容忍的偏差就越大。因此，在不同的尺度上，相同形状的曲线将具有相同的意义，即可以产生相似形状的多边形近似。

相比于给出固定误差阈值，本算法在生成高质量的通用多边形近似时，动态地选择最合适的误差阈值。例如，对于不同尺寸的曲线，算法可以给出相似形状的多边形近似，但比例不同。如果使用固定阈值，只能粗略地近似较短的曲线，并且在近似较长曲线时会添加不必要的细节。图 2-2 显示了检测自行车图像中线段的结果，其中车轮由多边形近似表示（描述中的顶点由十字表示）。

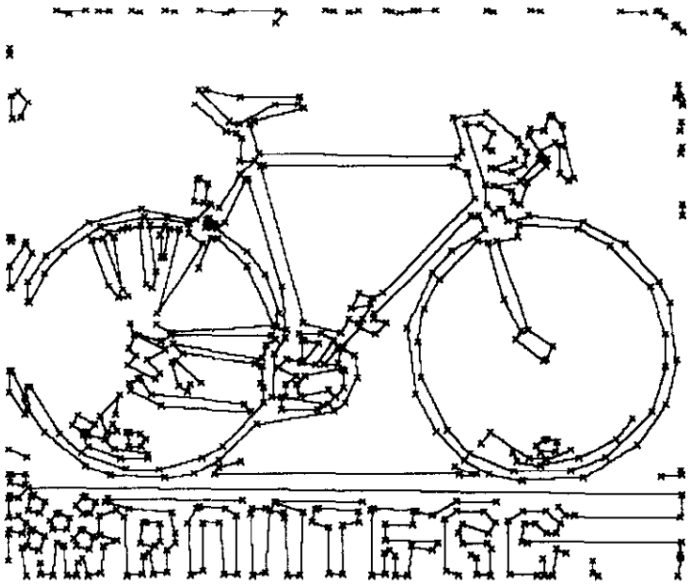


图 2-2 使用直线段近似图像中的曲线[6]

2.3 圆弧段检测

通过扩展Lowe[16]的工作，可以用相同的递归算法进行圆弧段检测。在直线段检测中，为了确定曲线路径如何用直线段描述，将散点序列假设为线段，并通过在最大偏差点处进行细分，递归地对这些被假设的线段进行重新排列。在圆弧段检测中，以类似的方式，将从多边形近似过程（即直线段检测过程）中获得的线段序列假设为曲线，并通过在最大偏差点处进行细分，在本工作中，把最大偏差定义为$d(deviation)$，高于此偏差的将使用递归二分法继续处理曲线，重新绘制一条用圆弧段表示的曲线路径，如图 2-3所示。需要注意的

是，这里的拟合结果是圆弧段与原始数据的拟合，只是为了简化在圆弧段检测过程中的计算，选择将圆弧直接拟合到在直线段检测中得到的直线段描述中。

图 2-3 将圆弧拟合到图 2 数据示例，该数据示例中指示了最大偏差点

相对于直线，圆弧具有更多的自由度，因此圆弧检测比直线检测更为复杂。圆弧的规范定义需要更多的参数来描述其几何特征。在将圆弧拟合到空间曲线时，需要确定圆心坐标、起点坐标、终点坐标以及半径长度等参数。这些参数的确定需要考虑到圆弧在曲线中的位置和形状等因素，因此圆弧拟合算法的设计更为复杂。

Landau[17]和 Albano[18]提出了通用圆弧拟合算法，该算法可以将圆弧拟合到任意空间曲线中。该算法使用非线性最小二乘法来迭代地优化圆心坐标、起点坐标、终点坐标和半径长度等参数，以最小化圆弧和曲线之间的距离误差。

为了简化圆弧拟合的过程，可以通过应用和在直线段检测中描述的用于拟合直线的约束相同的约束来减少自由度的数量。由于空间曲线是由连接在一起的直线段和圆弧段的组合来描述的，因此圆弧段被约束为在要进行拟合的空间曲线的两个端点处开始和结束。给定圆弧的两个端点，需要在二维空间中确定三个参数，即该圆弧对应的圆的半径长度、圆心的x坐标和圆心的y坐标。虽然最小化这三个参数误差的重要性程度较低，但由于圆弧的起点和终点是已知的，因此可以对问题进行进一步简化：由于圆心到圆弧起点的距离等于圆心到圆弧终点的距离，因此圆心被限制在一条直线上，该直线垂直于连接圆弧起点和终点的直线，如图 2-4 所示。

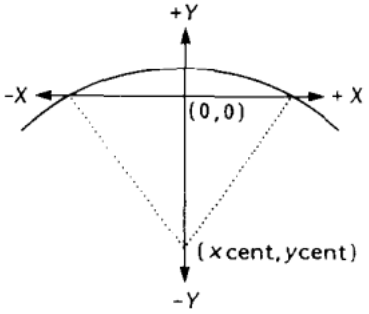


图 2-4 给定圆弧，圆心被限制在一条直线上

如公式(2-1)所示，在该算法中，使用最小二乘误差准则来确定原始数据和圆弧之间的匹配（误差）程度。

$$error_{tot} = \sum_{i=2}^{N-1} e_i^2 \tag{2-1}$$

误差总和被定义为所有原始数据顶点到圆弧段的最短距离的平方和，其中误差是半径长度和从原始数据顶点到圆心的距离之间的差值，如图 2-5所示。本文使用的递归二分法使用梯度下降方法快速搜索最小误差[19]。

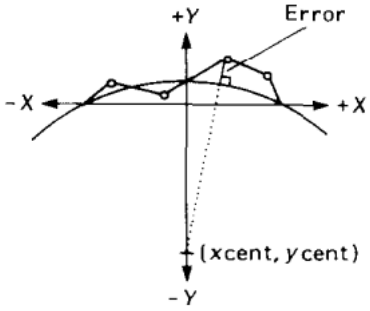


图 2-5 将圆弧拟合到多边形近似时的最大偏差

以下是在二维空间中计算圆弧段的各个参数的算法，该算法分为以下几个阶段。

首先，变换顶点的坐标，使连接两个原始数据顶点的线的中心位于原点，两个原始数据顶点将位于x轴上，这意味着圆心将位于y轴上。

其次，如公式(2-2)所示，计算假设圆心位于原点$(0,0)$以及x轴$(0,1)$和$(0,-1)$两侧的误差。这使得能够在原点处即可确定误差函数的梯度。所需的最小误差值将位于由负梯度的方向指示的x轴的一侧。

$$\begin{cases} direction = 1, & error(0,1) \le error(0,-1) \\ direction = -1, & error(0,1) > error(0,-1) \end{cases} \tag{2-2}$$

第三，计算所有顶点的平均y坐标。这表示弧在x轴的哪一侧。然后将此测

量与方向结合使用，使用以下规则来决定弧是小弧（弧对向角小于180度）还是大弧（弧对向角大于或等于180度）：

当$direction = 1$且$y_mean < 0.0$时，该弧为小弧

当$direction = -1$且$y_mean > 0.0$时，该弧为小弧

当$direction = 1$且$y_mean \geq 0.0$时，该弧为大弧

当$direction = -1$且$y_mean \leq 0.0$时，该弧为大弧

第四，搜索最小误差。在本文中，使用递归二分法来快速找到最小误差值。

第五，计算线段的“显著性”，即最大偏差与圆弧段长度的比值。

计算出圆弧段在二维空间中的弧度和圆心后，我们需要将这些参数映射到三维空间中。这是因为在三维空间中，圆弧段的位置信息不仅包括弧度和圆心，还包括圆弧段所在的平面和圆弧段的半径长度。因此，我们需要使用 CGAL 库中的映射函数将二维空间中计算出的参数映射到三维空间中对应的参数。

CGAL 库中的映射函数可以将二维平面上的点、向量或线段映射到三维空间中的点、向量或线段。在本文中，我们需要将圆弧段的弧度和圆心映射到三维空间中的参数。具体地，我们可以将圆弧段的圆心映射到三维空间中的一个点，将圆弧段所在的平面映射到三维空间中的一个平面。

在三维空间中，我们可以利用已知的弧度和半径长度来计算圆弧段的圆心坐标。具体地，我们可以使用公式(2-3)所示的向量运算来计算圆弧段的圆心坐标。其中假设圆弧段所在的平面的法向量为n，圆弧段的起点坐标为$P0$，圆弧段的半径长度为r，圆弧段的弧度为θ，$\cos\left(\frac{\theta}{2}\right)$是圆弧段的弦长与半径长度之比。

$$C = P0 + r * \cos\left(\frac{\theta}{2}\right) * n \tag{2-3}$$

因此，通过在二维空间中计算出圆弧段对应的弧度和圆心，再利用 CGAL 库中的映射函数将二维空间中计算出的参数映射到三维空间中对应的参数，最后利用已知的弧度和半径长度计算出圆弧段的圆心坐标，我们就可以确定圆弧段在三维空间中的位置信息。

算法 2-2 用伪代码形式展示了如何在二维空间中计算圆弧段的各个参数，以及如何将对应圆弧段映射到三维空间中。

算法 2-2　单条圆弧段检测算法

Single Arc Detection Algorithm

INPUT：*start_segment_index*: 直线段起点下标;
end_segment_index: 直线段中点下标

OUTPUT：arc_element: 将输入的直线段序列预设为一条圆弧段

1: **for** each point p **from** start_segment_index **to** end_segment_index **do**
2: 3Dpoints_projection(points,plane_curve)：拟合点到 3D 空间中的平面曲线上
3: plane_3D_to_2D(plane_Curve,plane_curve_2D)：将 3D 空间中的平面曲线转到 2D 空间中
4: SingleArcfitting2D(plane_curve_2D,center_2D,arcmid_2D)：2D 空间中计算出圆心和弧中点
5: arc_element.center=plane_2D_to_3d(center_2D)
6: arc_element.middle=plane_2D_to_3d(arcmid_2D)：利用 CGAL 库将 2D 空间中的圆心和弧中点转到 3D 空间中
7: arc_element.radius=Math::GetLength(arc_element.center-arc_element.middle)：在 3D 空间中计算出圆弧段对应的半径
8: return arc_element

该检测算法中的递归程序反复将弧细分为子弧，当要用圆弧段进行近似的线段集合包含少于三个线段时停止递归。这类似于折线近似中的三像素极限。显然，一条线段不应该被一条圆弧所取代。因为两条线段包含三个点，并且总是能被一段圆弧完美拟合，这将使每三个点（每两段直线段）就会被一条圆弧段拟合匹配，阻止了用更长的一条圆弧段拟合三段及以上数量的直线段，与我们初始的用尽量少的直线段和圆弧段拟合曲线的研究目的不符。

在进行圆弧拟合时，需要测量拟合结果与原始曲线之间的偏差。通常情况下，可以计算每个点到拟合圆弧的距离，然后返回最大的距离作为偏差测量。然而，这种方法可能会受到离群点的干扰，从而导致偏差测量结果不准确。为了解决这个问题，本文提出了一种通过绕弧步进的方法来计算偏差测量。具体来说，该方法将直线段上的每个点沿着拟合圆弧的弧长方向进行步进，计算每个点到圆弧的最小距离，并返回所有距离中的最大值作为偏差测量。由于该方法考虑了圆弧的弧长方向，因此可以更好地处理曲线中的离群点和噪声。

虽然该方法的计算量稍有增加，但是它可以提高偏差测量的稳健性和准确性，从而得到更好的拟合结果。该方法可以应用于多种圆弧拟合和曲线拟合任务中，特别是在需要处理复杂曲线和噪声环境下的情况下，可以提高算法的鲁棒性和可靠性，算法 2-3 用伪代码形式展示了整个圆弧段检测的过程，其中递归函数 SingleArcDectection()部分伪代码见算法 2-2。

算法 2-3　圆弧段检测算法

Arc Detection Algorithm

INPUT：*fitting_Segments*: 存储用直线段拟合多段线路径的拟合结果

OUTPUT：fitting_Elements: 存储用直线段和圆弧段拟合多段线路径的拟合结果

```
1:  if fitting_Segments.size()<3 then
2:      return fitting_Elements
3:  SingleArcDetection(start_segment_index,end_segment_index,arc_element)：先
    假设段列表的起点与终点连成一条大圆弧
4:  for each point p in fitting_Segments do
5:      d=arc_distance(arc_points[i].p,arc_element)：计算点到预设圆弧段的误差距
    离
6:      max_deviation=max(d,max_deviation)：找到最大偏差
7:  if max_deviation>固定偏差阈值 then
8:      在最大偏差点处分割成两部分，分别重复上述算法步骤
9:  else
10:     return fitting_Elements
```

第3章G1 连续优化

3.1 概述

G0 连续和 G1 连续是曲线加工中非常重要的概念。G0 连续是指两条曲线在连接处位置上连续，即端点重合，而 G1 连续则是指曲线在连接处不仅位置连续，还要求切线方向也连续，即相邻两条曲线在连接处的切线方向相同且夹角为零度。

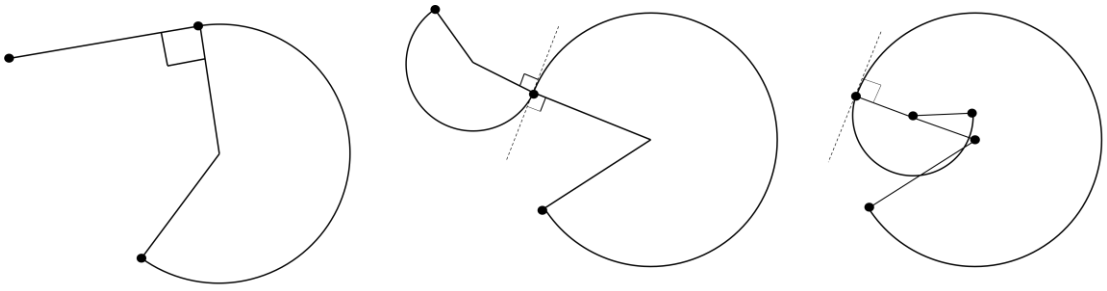
在机床加工中，G1 连续的加工路径非常重要，因为它可以确保相邻线段之间的平滑过渡，避免了加工过程中的突变和拐角，从而提高了加工精度和表面质量。如果加工路径不是 G1 连续的，会导致机床停顿或调整，降低加工效率，增加生产成本。在精密机械加工领域，G1 连续的加工路径也是保证加工精度和表面质量的重要因素之一[20]。

因此，掌握 G1 连续的曲线加工技术对于提高加工效率和质量具有重要意义。在实际生产中，必须充分考虑 G1 连续的要求，避免出现加工误差和表面残留的痕迹，从而提高加工质量，降低生产成本。

在实际曲线加工中，为了保证 G1 连续，需要对相邻的拟合段进行调整。一种常见的方法是通过调整圆弧的半径和圆心位置或者调整直线段的端点位置来实现。这样可以确保相邻线段在连接处的切线方向一致，从而实现 G1 连续。

3.2 G1 连续实现

本文中用于拟合三维散点路径的基元为直线段和圆弧段，直线段与直线段之间的 G1 连续只有一种情况，即两个直线段共线的情况，包含圆弧段的 G1 连续连接有三种情况，如图 3-1 所示，(a)是指直线段与圆弧段连接的情况，图(b)是指连接弧与已知弧外切的情况，图(c)连接弧与已知弧内切。



(a) 直线段与圆弧段连接　　(b) 连接弧与已知弧外切　　(c) 连接弧与已知弧内切

图 3-1 圆弧段 G1 连续连接的三种情况

在曲线加工中，为了实现 G1 连续的曲线加工路径，可以通过增加过渡弧来连接相邻的线段。这样的过渡弧称为圆弧插补曲线[21]，它可以确保曲线加工路径在相邻线段之间平滑转换，避免出现拐角或突变。但是在本文中，由于部分散点是直接用圆弧段进行的拟合，如果再使用圆弧对曲线进行插补平滑连接，会对视觉效果造成一定影响，并且使用圆弧插补平滑曲线所需要的计算量也稍大。

因此，在本文中，为了保持相邻两段的切线方向一致，并减少对视觉效果的影响以及计算量的增加，在连接相邻的直线段和圆弧段时，我们使用了数学计算直接对直线段和圆弧段的位置进行调整。具体来说，通过计算相邻线段之间的切线方向，调整圆弧段的圆心位置和半径，或者调整直线段的端点位置，以确保相邻两段的切线方向一致，从而实现 G1 连续的曲线加工路径。

尽管使用圆弧插补曲线可以更好地实现曲线加工路径的平滑过渡，但在本文的情况下，使用数学计算直接对直线段和圆弧段的位置进行调整，不仅可以保证 G1 连续，还可以减少计算量，并且可以更好地控制曲线的视觉效果。因此，这种方法非常适合用于本文所涉及的曲线拟合和加工任务。

该实现分为以下几个步骤。

首先，计算每个拟合段的长度和切线方向。在基元检测中，已经建立了包含所有直线段和圆弧段的列表，遍历各段，对于直线段，用相邻两点坐标计算斜率即可，对于圆弧段，根据切线与半径垂直的特点，将切线斜率定义为k_1，将半径所在直线斜率定义为k_2，根据公式(3-1)可计算出圆弧端点处切线的斜率。

$$k_1 * k_2 = -1 \tag{3-1}$$

其次，检查相邻拟合段的切线方向是否一致。如果相邻拟合段的切线方向不一致，即它们之间存在一个夹角，我们需要重新计算圆心坐标以保持曲线的连续性。这是因为，如果我们使用两个不同的圆心来拟合相邻的拟合段，那么它们之

间的连接点将不平滑，导致曲线的不连续性。

为了重新计算圆心坐标$(x,\ y)$，我们可以使用公式(3-2)进行计算。

$$\begin{cases} x = x_0 + r * \cos\left(\theta_0 + \frac{\theta}{2}\right) \\ y = y_0 + r * sin\left(\theta_0 + \frac{\theta}{2}\right) \end{cases} \tag{3-2}$$

其中x_0, y_0是当前段终点的坐标，θ_0是当前段的角度值，若当前段是直线段，则θ_0即为当前段与x轴所成的夹角，若当前段是圆弧段，则θ_0为圆弧段的弧度值，θ为当前段终点处的切线和连接段起点处的切线所成的夹角。

公式(3-2)中的$r * \cos\left(\theta_0 + \frac{\theta}{2}\right)$和$r * \sin\left(\theta_0 + \frac{\theta}{2}\right)$分别表示圆心到当前段终点处的向量在$x$轴和$y$轴上的投影，其中$r$为圆弧段的半径，$\theta$为当前段终点处的切线和连接段起点处的切线所成的夹角。通过计算出圆心坐标(x, y)，我们就可以重新生成圆弧段的位置，从而保持相邻两段的切线方向一致，保证 G1 连续性。

第三，在调整相邻拟合段的连接处时，对于圆弧段，需要计算出调整后的圆心坐标和半径，然后根据参数方程重新生成圆弧段位置。具体来说，通过公式(3-2)计算出圆心坐标(x, y)，然后根据半径r和公式(3-3)所示的圆弧的参数方程来重新生成圆弧段的位置。这样可以保证相邻两段的切线方向一致，从而保证 G1 连续性。对于直线段，直接调整起点与终点的位置可以保证相邻两段的连接处切线连续。具体来说，在连接相邻的直线段时，通过计算相邻线段之间的切线方向，调整直线段的端点位置，以确保相邻两段的切线方向一致，从而实现 G1 连续的曲线加工路径。

$$\begin{cases} x = x_0 + r * \cos\theta \\ y = y_0 + r * sin\theta \end{cases} \tag{3-3}$$

对所有相邻的拟合段重复以上步骤，直到所有的相邻拟合段切线方向一致，G1 连续性满足。将所有拟合好的直线段和圆弧按顺序连接起来，形成最终的拟合曲线。

通过以上算法，我们可以实现用直线段和圆弧拟合散点路径并保证 G1 连续。该算法简单易懂，且计算量较小，非常适合在计算机辅助设计和制造领域中使用，例如在 CAD、CAM 和机器人控制等方面。使用这种方法可以大大提高拟合效率和精度，并且可以得到平滑的曲线加工路径，从而提高加工质量和效率。

第4章 算法评价

4.1 运行环境及参数

在本工作中，主体算法实现部分使用了 C++语言完成，以及基于 Qt 平台，使用了 CGAL 库和 Open CASCADE 完成了直线段检测和圆弧段检测算法，基于 Python 语言，使用 PyVista 库完成了可视化部分。

程序运行的硬件配置为 CPU:11th Gen Intel(R) Core(TM) i7-11700F @ 2.50GHz，内存:32GB，GPU 型号为 NVIDIA GeForce RTX 3060 Ti。软件配置为：操作系统： Windows 10。

与算法评价相关的参数有三个，分别是固定偏差阈值(deviation)、拟合使用的直线段数量和圆弧段数量，不同固定偏差阈值的设定会影响拟合使用的直线段数量和圆弧段数量。理论上来说，固定偏差阈值越大，所能容忍的动态最大偏差值越大，散点路径拟合用到的直线段和圆弧段的数目越少，但也会在一定程度上影响拟合效果；反之，固定偏差阈值越小，所能容忍的动态最大偏差值越小，散点路径拟合用到的直线段和圆弧段的数目越多，拟合出的曲线越贴近原始数据。故我们在进行参数调优时的目标就是找到合适的固定偏差阈值，使得在使用尽可能少的直线段和圆弧段的情况下，达到尽可能好的拟合效果。

4.2 不同最大偏差值对结果外观效果的影响

根据表 4-1 所示的数据，我们可以发现固定偏差阈值对模型拟合的时间和精度都具有显著的影响。随着固定偏差阈值的增大，模型的处理时间逐渐减少，而拟合模型所使用的线段数量也逐渐减少。这是因为在递归二分法实现的散点拟合算法中，较大的固定偏差阈值可以容忍更大的动态偏差阈值，从而减少递归迭代次数，降低处理时间和线段数量。

然而，设置过大的固定偏差阈值可能会导致模型的精度下降。在固定偏差阈值较小时，模型的处理时间和线段数量相对较大，因为算法需要更多的迭代次数才能满足精度要求。但是，较小的固定偏差阈值可以提高模型的拟合精度，因为算法需要更多的迭代次数来逐步逼近散点的真实曲线。

表 4-1 还显示了固定偏差阈值在 0.5 到 2.5 范围内对同一模型的处理时间和线段数量的影响明显，而在固定偏差阈值为 2.5 和 3.5 时，处理时间和线段数量相对稳定，这说明算法实现过程中计算的动态偏差阈值主要集中在 2.5 左右或小于 2.5 的范围内。这进一步验证了本文使用的拟合算法在精度和效率上都是相对可靠的。因此，在实际应用中，可以根据需要选择适当的固定偏差阈值来平衡模型的拟合精度和处理效率。

表 4-1 不同模型在不同偏差值下对应的处理效果

	D1：0.5		D2:1.5		D3：2.5		D4：3.5	
	处理时间	段数	处理时间	段数	处理时间	段数	处理时间	段数
模型 1	14.39s	148	7.56s	76	5.90s	50	5.86s	45
模型 2	44.85s	456	23.58s	235	16.52s	160	13.36s	129
模型 3	2.73s	88	1.41s	42	1.03s	28	0.88s	24

4.3 结果展示

图 4-1、图 4-3、图 4-5 分别是三个模型的原始输入数据可视化，相邻小线段之间用不同颜色进行了标注，可见原始模型输入是由多个离散的小线段连接而成的曲线。

图 4-2、图 4-4、图 4-6 分别是针对三个模型在不同固定偏差阈值下的拟合结果可视化。从左到右，从上到下分别对应固定偏差阈值为 0.5、1.5、2.5、3.5 时的拟合结果。我们可以观察到，随着固定偏差阈值的增大，拟合模型散点使用的线段数量逐渐减少，其中绿色标注的是圆弧段，黄色标注的是直线段，相邻段的连接点处用灰色箭头标注了连接点的法向量方向。

在视觉效果上，图中的结果与本文 4.2 章节中的结果分析相符。然而，我们需要注意到在模型 2 的拟合结果中，并未很好地完成圆弧段的拟合。这可以归因于原始数据存在较多噪声，而本项目研究中缺乏对原始数据进行噪声处理的方法。

在对比模型 2 和模型 3 时，我们发现模型 3 的原始数据只有 506 个散点，而模型 2 的原始数据则多达 1463 个散点。由于散点数量越大，存在噪声的点也越多，因此在图 4.2 中我们也可以看到存在多处连续的呈波浪状的圆弧段。虽然

可以通过增大固定偏差阈值将多个小段拟合成一条较长的圆弧段，但这会对拟合精度产生较为严重的影响。因此，在未来的工作中，我们需要更好地处理数据噪声，例如采用高斯平滑等方法以提高模型的拟合精度和视觉效果。

图 4-1 模型 1 的原始输入数据可视化

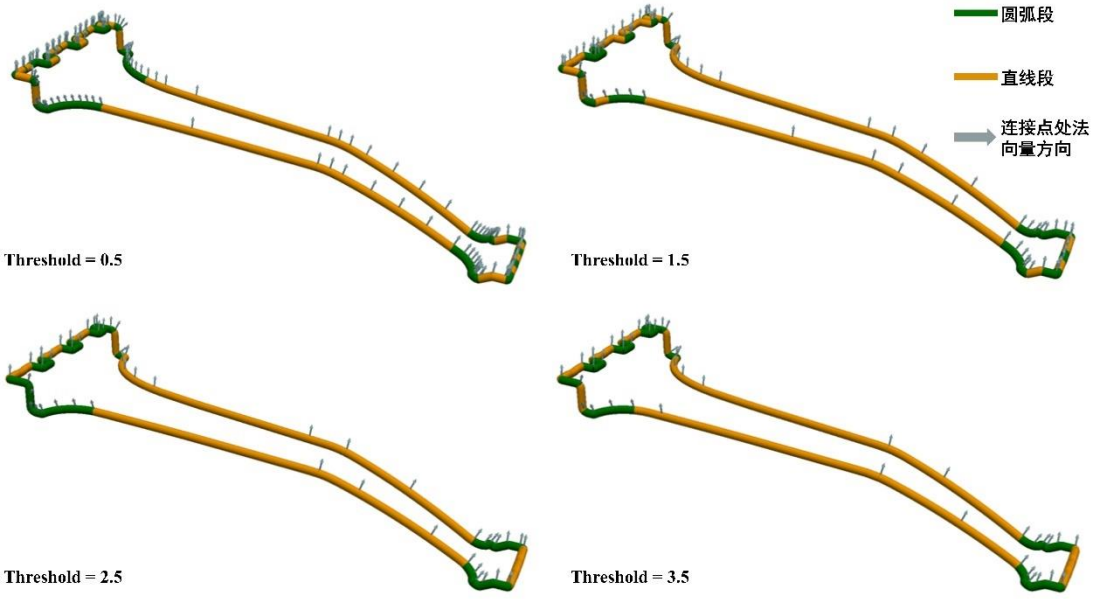


图 4-2 模型 1 在不同固定偏差阈值下的拟合结果可视化

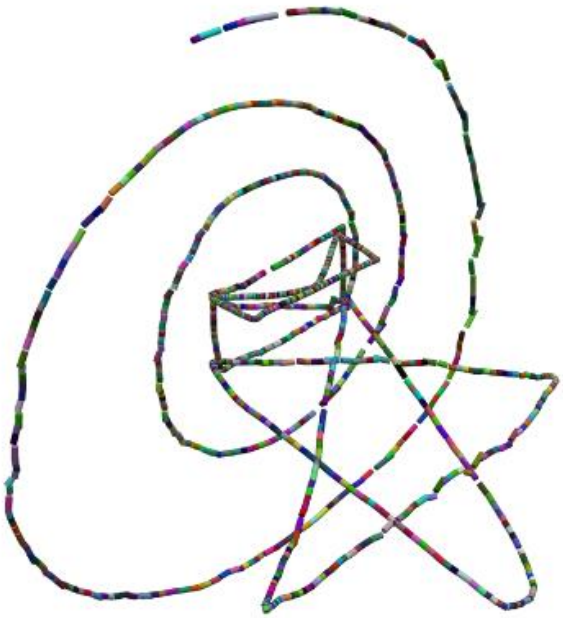


图 4-3 模型 2 的原始输入数据可视化

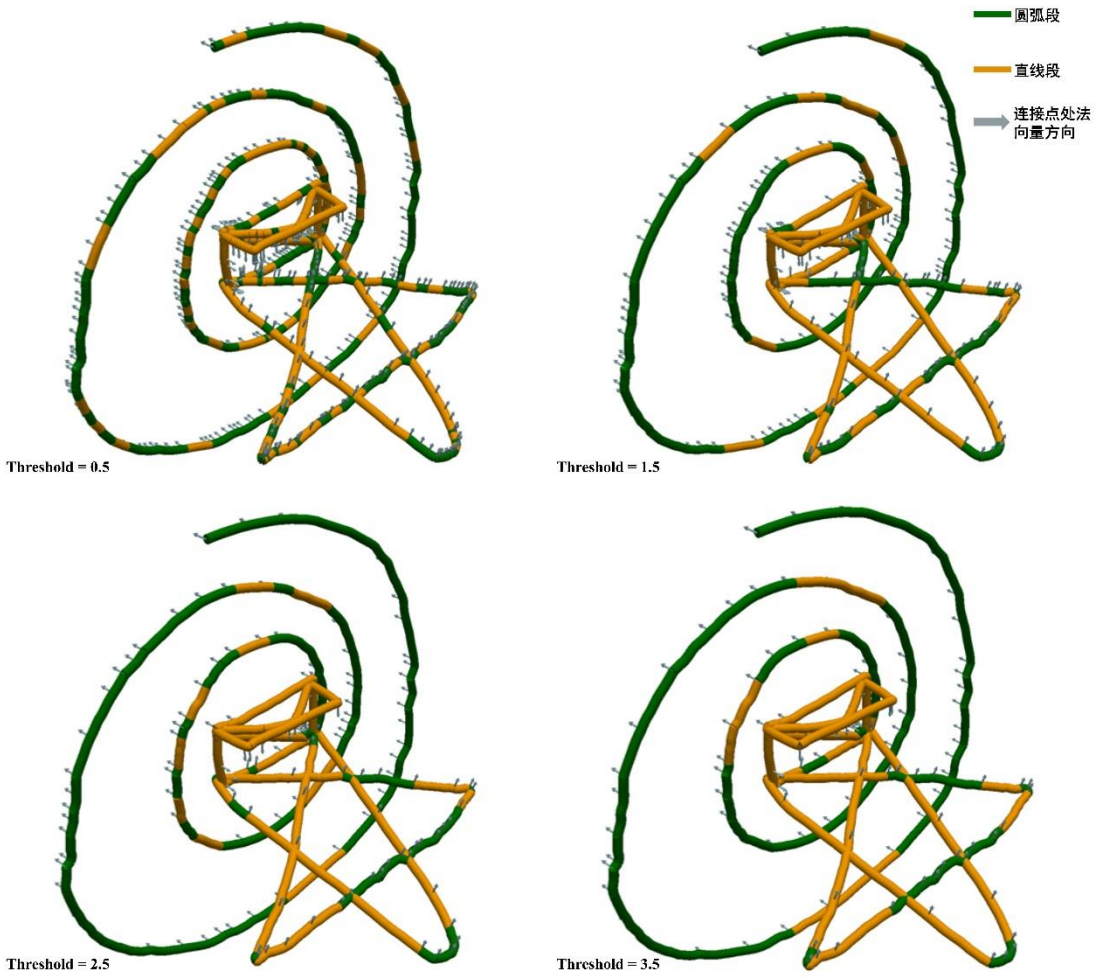


图 4-4 模型 2 在不同固定偏差阈值下的拟合结果可视化

图 4-5 模型 3 的原始输入数据可视化

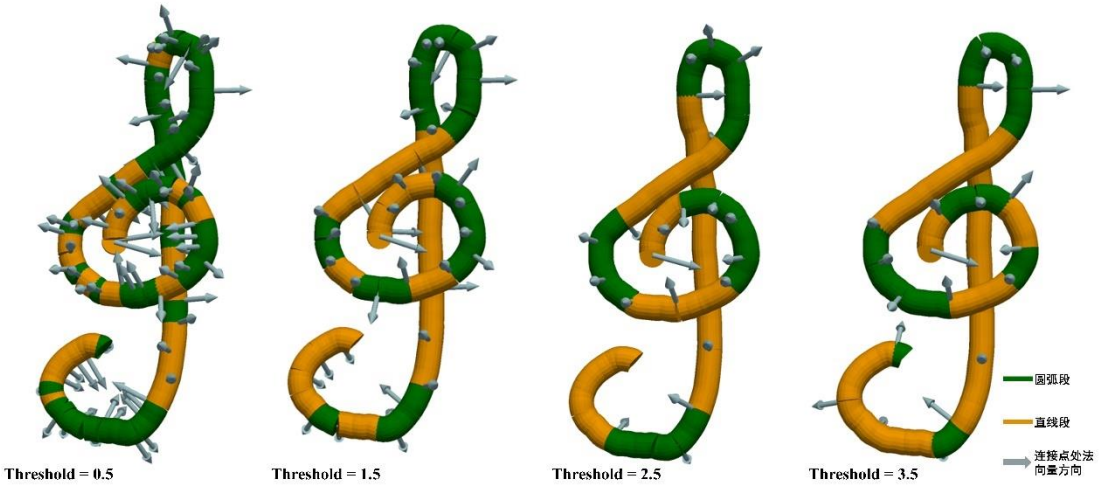


图 4-6 模型 3 在不同固定偏差阈值下的拟合结果可视化

第5章 总结与展望

5.1 本文工作总结

5.1.1 主要贡献点

本工作针对多段线路径的直线段和圆弧 G1 连续拟合问题，提出了一种通用算法，该算法扩展了 Lowe 提出的直线拟合算法，将由多个连续的小线段组成的曲线路径描述为直线和圆弧的组合。这种算法具有重要的理论意义和实际应用价值。本工作的主要贡献点包括以下几个方面：

1. 提出了一种新的算法，该算法能够解决面向多段线路径的直线段和圆弧 G1 连续拟合问题，实现更加精确和高效的路径拟合。该算法为工业设计、机器人和计算机图形学等领域提供了更加优秀的技术支持。

2. 本算法使用了一种快速算法来确定圆弧拟合段与原始多段线数据的最小误差匹配。相对于其他曲线拟合方法，本方法使用了曲线必须由两个定义的端点定界这一约束，减少了曲线位置参数的计算负担，提高了算法的运算效率。

3. 该工作还可以扩展到使用其他曲线类型来对多段线路径进行拟合。例如，可以使用椭圆取代圆弧进行拟合。这种方法可以扩展该算法的适用范围，使其更加灵活和多样化。

该算法的一个有趣的特性是，在曲线拟合的每个阶段，都会动态选择“最佳”的阈值，从直线检测到圆弧检测，在任何阶段都不需要预设固定的阈值。这种动态阈值的选择使该算法具有通用性，能够适应不同的数据集和应用场景。

5.1.2 存在的问题

在实现 G1 连续性时，针对切线导数不连续的拐角点进行平滑处理是非常重要的。然而，在本文中，我们并未对每个连接点处的切线倒数是否连续进行区分，而是将所有连接点都视为拐角点，这导致了平滑过程中出现了不必要的计算量，并且对平滑效果产生了负面影响。

具体来说，由于我们没有预先对切线倒数不连续的拐角点进行自适应平滑处理，因此在实现 G1 连续性时，平滑处理过程会对所有连接点进行操作，而不仅仅是切线倒数不连续的拐角点。

如果我们能够预先对切线倒数不连续的拐角点进行自适应平滑处理，就可以避免对所有连接点进行操作。这样可以有效降低计算量，并提高平滑效果。因此，在未来的研究中，我们可以考虑引入自适应平滑处理来提高算法的效率和准确性。

总之，本文的方法虽然可以实现 G1 连续性，但是在平滑处理中存在一些不足之处。在未来的研究中，我们可以进一步完善该方法，以提高平滑效果和算法的效率。

5.1.3 总结

综上所述，本研究的选题背景和研究意义在于解决当前数控机床设备在只能使用通用控制直线段和圆弧的指令集，而无法应用高级曲线拟合技术的情况下，如何进行高效率加工的问题。该算法具有广泛的应用前景，将为工业生产和计算机辅助制造的发展产生积极影响。

5.2 未来工作

本文的研究工作仍然有许多问题值得进一步研究探索，具体包括以下几个方面。

5.2.1 连续性平滑优化

未来工作可以通过首先将沿曲线的切线方向上应用自适应平滑，然后取平滑切线方向的导数，最后在导数上定位高尖峰，来检测拐角。平滑连接可以首先通过动态聚焦拟合技术粗略定位，然后通过调整算法进行细化。进一步使用动态聚焦拟合技术将曲线段（由一对相邻的角限定）的一端固定，并从另一端开始对曲线段进行扫描，直到它聚焦在拟合直线或圆弧的组成部分上。识别该组成部分，并以相同的方式对曲线的其余部分重复该过程。最后在细化阶段，逐点向左或向右调整每个平滑连接，直到优化曲线段的拟合优度测量。除了 G1 连续性，未来工作可以考虑研究更高阶的 G 连续性，例如 G2 或 G3。这将使得拟合的曲线更加平滑，更接近原始数据。例如，在本文中是保持切线连续实现了 G1 连续性，对于 G2 连续性，可以在拟合过程中保持曲率连续性，以获得更平滑的拟合曲线。

5.2.2 研究采样点密度和噪声对拟合结果的影响

在实际应用中，多段线路径可能存在噪声和异常值，这会影响拟合结果的准

确性。此外，采样点密度也会影响拟合结果的好坏。因此，如何使用滤波算法或异常值检测算法来提高拟合结果的准确性，以及如何在不同密度下获得最佳结果，是未来工作的一个重要方向。

一种常见的滤波算法是高斯滤波算法，它可以有效地去除噪声和异常值。使用高斯滤波算法时，需要选择合适的卷积核大小和滤波强度，以平衡去除噪声和保留曲线特征之间的关系。

异常值检测算法可以通过识别和剔除异常点来提高拟合结果的准确性。常用的异常值检测算法包括基于统计学的算法和基于距离的算法。在使用异常值检测算法时，需要根据实际情况选择合适的算法和参数，以确保检测到的异常值不会误删真实的曲线特征。

在不同密度下获得最佳结果的方法之一是使用自适应采样方法。自适应采样方法可以根据曲率变化自动调整采样密度，从而在不同密度下获得最佳结果。具体来说，可以根据曲率变化率来调整采样密度，使得曲率变化率较大的地方采样密度较高，曲率变化率较小的地方采样密度较低。

综上所述，通过使用滤波算法、异常值检测算法和自适应采样方法，可以提高拟合结果的准确性，并在不同密度下获得最佳结果。这些方法可以在实际应用中被广泛使用，从而提高拟合效率和精度，实现平滑的曲线加工路径和轨迹，从而提高加工质量和效率。

5.2.3 实际场景应用

本文提出的用直线段和圆弧拟合多段线路径并保证 G1 连续的算法，是一种非常实用和有效的方法。该方法可以应用于各种实际场景，例如数控加工、路径规划、自动驾驶等领域。遗憾的是，在本研究中未实现将路径拟合算法应用到实际的机床刀具加工中，在未来工作中可以探索如何将该方法应用于数控加工、路径规划、自动驾驶等领域，并进行相关的实验和评估。

在数控加工领域中，该方法可以用于生成平滑的刀具路径，从而提高加工质量和效率。在路径规划领域中，该方法可以用于生成平滑的路径，从而提高路径跟踪的精度和稳定性。在自动驾驶领域中，该方法可以用于生成平滑的轨迹，从而提高车辆驾驶的安全性和舒适性。

为了将该方法应用到实际场景中，需要进行相关的实验和评估。具体来说，

可以将该方法应用于数控加工中，生成不同形状的刀具路径，并与传统的拟合方法进行比较，评估该方法的精度和效率。在路径规划和自动驾驶领域，可以将该方法应用于实际的车辆控制中，评估其路径跟踪精度和稳定性，并与传统的路径规划方法进行比较。

总之，将该方法应用到实际场景中，可以极大地提高拟合效率和精度，实现平滑的曲线加工路径和轨迹，并且可以在各种领域中发挥重要的作用。

致　　谢

行文至此落笔中，始于初冬终于夏。

时间飞逝，如白驹过隙。大学的学习生活即将落下尾声，在这四年的学习生活中，我收获了很多，也成长了很多，经历了挫折，也获得了成绩，而这些成绩的取得是和一直关心帮助我的人分不开的。

本次毕业设计是对我大学四年学习下来最好的检验。经过这次毕业设计，我的能力有了很大的提高，比如操作能力、分析问题的能力、合作精神、严谨的工作作风等方方面面都有很大的进步。

这期间凝聚了很多人的心血，在此我表示由衷的感谢。没有他们的帮助，我将无法顺利完成这次设计。首先，我要特别感谢我的指导老师——赵海森老师，对我的悉心指导，在我的论文书写及设计过程中给了我大量的帮助和指导，为我理清了设计思路和操作方法，并对我所做的课题提出了有效的改进方案。赵海森老师渊博的知识、严谨的作风和诲人不倦的态度给我留下了深刻的印象。从他身上，我学到了许多能受益终生的东西。再次对赵海森老师表示衷心的感谢，桃李不言，下自成蹊，言辞有尽，师恩永记！

山水一程，三生有幸。感谢一直陪伴在我身边的同学朋友们。风雨同舟，苦乐共济，感恩相遇，未来可期！

参考文献

[1] Li M, Zhang H. AUV 3D path planning based on A* algorithm[C]//2020 Chinese Automation Congress (CAC). IEEE, 2020: 11-16.

2 Yang Z, Shen L Y, Yuan C M, et al. Curve fitting and optimal interpolation for CNC machining under confined error using quadratic B-splines[J]. Computer-Aided Design, 2015, 66: 62-72.

[3] Yang L, Qi J, Song D, et al. Survey of robot 3D path planning algorithms[J]. Journal of Control Science and Engineering, 2016, 2016.

[4] Wu Z, Tucker T M, Nath C, et al. Step ring based 3D path planning via GPU simulation for subtractive 3D printing[C]//International Manufacturing Science and Engineering Conference. American Society of Mechanical Engineers, 2016, 49903: V002T04A006.

[5] 李浩，吴文江，韩文业,等. 基于公差带的二次连续贝塞尔曲线刀具轨迹平滑算法[J]. 吉林大学学报：工学版, 2019, 49(2):9.

[6] Rosin P L， West G A . Segmentation of edges into lines and arcs[J]. Image & Vision Computing, 1989, 7(2):109-114.

[7] Marsala M, Mantzaflaris A, Mourrain B. Point Cloud Fitting by G1 Smooth Spline Functions[J]. 2023.

[8] Pavlidis T， Horowitz S L . Segmentation of Plane Curves[J]. IEEE Transactions on Computers, 1974, C-23(8):860-870.

[9] Freeman H， Davis L S . A Corner-Finding Algorithm for Chain-Coded Curves[J]. IEEE Transactions on Computers, 1977, C-26(3):297-303.

[10] Pridmore A P， Porrill J， Mayhew J . Segmentation and description of binocularly viewed contours[J]. Butterworth-Heinemann, 1987, 5(2):132-138.

[11] Gillard J . Circular and Linear Regression: Fitting Circles and Lines by Least Squares[J]. Journal of the Royal Statistical Society, 2011, 174(3):843-843.

[12] Yeung M K, Walton D J. Curve fitting with arc splines for NC toolpath generation[J]. Computer-Aided Design, 1994, 26(11): 845-849.

[13] West G， Rosin P L . Techniques for segmenting image curves into meaningful descriptions[J]. Pattern Recognition, 1991, 24(7):643-652.

[14] Wan W， Ventura J A . Segmentation of Planar Curves into Straight-Line Segments and Elliptical Arcs[J]. Graphical Models & Image Processing, 1997, 59(6):484-494.

[15] Bertino A, Bagheri M, Krstić M, et al. Experimental autonomous deep learning-based 3d path planning for a 7-dof robot manipulator[C]//Dynamic Systems and Control Conference. American Society of Mechanical Engineers, 2019, 59155: V002T14A002.

[16] Lowe D G . Three-dimensional object recognition from single two-dimensional images[J]. Artificial Intelligence, 1987, 31(3):355-395.

[17] Landau U M . Estimation of a circular arc center and its radius[J]. Computer Vision Graphics & Image Processing, 1987, 38(3):317-326.

[18] Antonio, Albano. Representation of Digitized Contours in Terms of Conic Arcs and Straight-Line Segments[J]. Computer Graphics & Image Processing, 1974.

[19] Ruder S. An overview of gradient descent optimization algorithms[J]. arXiv preprint arXiv:1609.04747, 2016.

[20] Beudaert X, Pechard P Y, Tournier C. 5-Axis tool path smoothing based on drive constraints[J]. International Journal of Machine Tools and Manufacture, 2011, 51(12): 958-965.

[21] 刘放，汪鎏，胡俊，等. 基于四元数的空间圆弧插补算法[J]. 机械科学与技术, 2009 (11): 1425-1428.

附录 1 代码及相关附件

2D 单个弧拟合

```
void Fitting::SingleArcFitting2D(std::vector<Vector2d> &curve, Vector2d &center_2d,
Vector2d &middle_2d)
{
    //OutputPlaneCurve before transformation
    #pragma region OutputPlaneCurve
    if (DebugInformation())
    {
        CGAL_Export_Path_Point output =
(CGAL_Export_Path_Point)GetProcAddress(hModule, "CGAL_Export_Path_Point");
        std::ofstream file("Debug\\plane_curve_before_transformation.obj");
        int index = 1;
        for (int i = 0; i < curve.size(); i++)
            output(file, index, "plane_curve_"+Math::IntString(i), 0.0, 0.0, 1.0,
Vector3d(curve[i][0], 0.0, curve[i][1]), 0.8);
        file.clear();
        file.close();
    }
    #pragma endregion

    std::vector<Vector2d> space_curve;
    //transformation
    ////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
    Vector2d curve_origin = (curve[0] + curve[curve.size() - 1]) /(float) 2.0;
    Vector2d curve_end_start = curve[curve.size() - 1] - curve[0];
    double curve_radius = Math::GetLength(curve_end_start) / 2.0;
    double curve_angle = Math::GetAngleBetween(curve_end_start, Vector2d(1.0, 0.0));
    double scale = 10.0 / curve_radius;
    for (int i = 0; i < curve.size(); i++)
    {
        Vector2d space_point = curve[i] - curve_origin;
        if (curve_end_start[1] >= 0)
            space_point = Math::RotationAxis2d(space_point, curve_angle, Vector2d(0.0,
0.0));
        else
            space_point = Math::RotationAxis2d(space_point, -curve_angle,
Vector2d(0.0, 0.0));
        space_point = space_point*(float)scale;
        space_curve.push_back(space_point);
    }
```

```
    ////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////

    //OutputPlaneCurve after transformation
    #pragma region OutputPlaneCurve
    if (DebugInformation())
    {
        CGAL_Export_Path_Segment output =
(CGAL_Export_Path_Segment)GetProcAddress(hModule, "CGAL_Export_Path_Segment");
        std::ofstream file("Debug\\plane_curve_after_transformation.obj");
        int index = 1;
        for (int i = 0; i < space_curve.size()-1; i++)
            output(file, index, "plane_curve_" + Math::IntString(i), 0.0, 0.0, 1.0,
            Vector3d(space_curve[i][0], 0.0, space_curve[i][1]),
Vector3d(space_curve[i+1][0], 0.0, space_curve[i+1][1]), 0.08);
        file.clear();
        file.close();
    }
    #pragma endregion

    double minimal_y;
    double final_radius;
    GradientDescent(space_curve,minimal_y,final_radius);


    //middle_2d
    middle_2d[0] = 0.0;
    middle_2d[1] = minimal_y + final_radius;
    middle_2d = middle_2d / (float)scale;
    if (curve_end_start[1] >= 0)
        middle_2d = Math::RotationAxis2d(middle_2d, -curve_angle, Vector2d(0.0, 0.0));
    else
        middle_2d = Math::RotationAxis2d(middle_2d, curve_angle, Vector2d(0.0, 0.0));
    middle_2d = middle_2d + curve_origin;


    //transformation
    center_2d[0] = 0.0;
    center_2d[1] = minimal_y;
    center_2d = center_2d / (float)scale;
    if (curve_end_start[1] >= 0)
        center_2d = Math::RotationAxis2d(center_2d, -curve_angle, Vector2d(0.0, 0.0));
    else
        center_2d = Math::RotationAxis2d(center_2d, curve_angle, Vector2d(0.0, 0.0));
```

```
    center_2d = center_2d + curve_origin;




}
```

3D 单个弧拟合

```
bool Fitting::SingleArcFitting3D(int start_segment_index, int end_segment_index,
FittingElement &arc_element)
{
    //get points
    std::vector<Vector3d> points;
    //points.push_back(fitting_segments[start_segment_index].s->p);
    //for (int i = start_segment_index; i <= end_segment_index; i++)
    //   points.push_back(fitting_segments[i].e->p);

    for (int i = fitting_segments[start_segment_index].s->index; i <=
fitting_segments[end_segment_index].e->index; i++)
    {
        points.push_back(curve_points[i].p);
    }

    /************************************/
    if (false)
    {
        IGESControl_Controller::Init();
        IGESControl_Writer ICW("MM", 0);
        //creates a writer object for writing in Face mode with  millimeters
        TopoDS_Shape sh = OCC::MakeWires(points);
        ICW.AddShape(sh);
        //adds shape sh to IGES model
        ICW.ComputeModel();
        Standard_Boolean OK = ICW.Write("D:\\in.igs");
    }
    /************************************/

    //plane fitting
    //Constraint(*): The first and final point of input curve must be on the fitting
plane
    Vector3d plane_p, plane_n;
    if (!PlaneFitting(points, plane_p, plane_n))
```

```
        return false;//fail to fit single arc

    //OutputFittingPlane
    #pragma region OutputFittingPlane
    if (DebugInformation())
    {
        CGAL_Export_Path_Segment output =
(CGAL_Export_Path_Segment)GetProcAddress(hModule, "CGAL_Export_Path_Segment");
        std::ofstream file("Debug\\plane.obj");
        int index = 1;

        CGAL_Vector_Base base = (CGAL_Vector_Base)GetProcAddress(hModule,
"CGAL_Vector_Base");
        Vector3d base_v = base(plane_n);

        Math::SetVectorLength(base_v, 100.0);
        for (double angle = 0.0; angle < 360.0; angle = angle + 10.0)
        {
            double pi_angle = angle / 180.0*Math::Math_PI;
            Vector3d p = Math::RotationAxis(base_v, pi_angle, plane_n) + plane_p;
            output(file, index, "plane_curve", 0.0, 0.0, 1.0, plane_p, p, 0.8);
        }
        file.clear();
        file.close();
    }
    #pragma endregion

    //project curve onto the fitting plane
    std::vector<Vector3d> plane_curve;
    CGAL_3D_Plane_Points_Projection projection =
(CGAL_3D_Plane_Points_Projection)GetProcAddress(hModule,
"CGAL_3D_Plane_Points_Projection");
    projection(plane_p, plane_n, points, plane_curve);

    //transform plane_curve to 2d space
    std::vector<Vector2d> plane_curve_2d;
    CGAL_3D_Plane_3D_to_2D_Points plane_3D_to_2d =
(CGAL_3D_Plane_3D_to_2D_Points)GetProcAddress(hModule,
"CGAL_3D_Plane_3D_to_2D_Points");
    plane_3D_to_2d(plane_p, plane_n, plane_curve, plane_curve_2d);


    //single arc fitting in 2d space
    Vector2d center_2d, middle_2d;
```

```
    SingleArcFitting2D(plane_curve_2d, center_2d, middle_2d);

    CGAL_3D_Plane_2D_to_3D_Point plane_2D_to_3d =
(CGAL_3D_Plane_2D_to_3D_Point)GetProcAddress(hModule, "CGAL_3D_Plane_2D_to_3D_Point");

    arc_element.center = plane_2D_to_3d(plane_p, plane_n, center_2d);
    arc_element.middle = plane_2D_to_3d(plane_p, plane_n, middle_2d);
    arc_element.type = "arc";
    arc_element.s = fitting_segments[start_segment_index].s;
    arc_element.e = fitting_segments[end_segment_index].e;

    arc_element.radius = Math::GetLength(arc_element.center - arc_element.middle);

    if (false)
    {
        IGESControl_Controller::Init();
        IGESControl_Writer ICW("MM", 0);
        //creates a writer object for writing in Face mode with  millimeters
        TopoDS_Shape sh = OCC::MakeArc(arc_element.s->p, arc_element.middle,
arc_element.e->p);
        ICW.AddShape(sh);
        //adds shape sh to IGES model
        ICW.ComputeModel();
        Standard_Boolean OK = ICW.Write("D:\\out.igs");
    }

    return true;
}
```

3D 递归弧拟合

```
std::vector<FittingElement> Fitting::ArcsFitting3D(int start_segment_index, int
end_segment_index)
{
    std::cout << start_segment_index << " " << end_segment_index << std::endl;
    std::vector<FittingElement> fitting_elements_3d;

    //minimal segment number while arc fitting
    if (end_segment_index - start_segment_index < 1)
    {
        for (int i = start_segment_index; i <= end_segment_index; i++)
            fitting_elements_3d.push_back(fitting_segments[i]);
        return fitting_elements_3d;
    }
```

```
    //fitting an arc
    FittingElement arc_element;
    SingleArcFitting3D(start_segment_index, end_segment_index, arc_element);

    //compute the maximal deviation
    double max_deviation = -1000.0;
    int index = -1;

    for (int i = start_segment_index; i < end_segment_index; i++)
    {
        for (int j = fitting_segments[i].s->index + 1; j <=
fitting_segments[i].e->index; j++)
        {
            double distance = Distance2ArcElement(arc_element, curve_points[j].p);
            if (distance > max_deviation)
            {
                max_deviation = distance;
                index = i;
            }
        }
    }

    //for (int i = start_segment_index; i < end_segment_index; i++)
    //{
    //   double distance = Distance2ArcElement(arc_element, fitting_segments[i].e->p);
    //   if (distance > max_deviation)
    //   {
    //       max_deviation = distance;
    //       index = i;
    //   }
    //}

    //post process
    if (max_deviation > position_deviation)
    {
        double max_curvature = -1000.0;
        index = -1;
        for (int i = start_segment_index; i < end_segment_index; i++)
        {
            for (int j = fitting_segments[i].s->index + 1; j <=
fitting_segments[i].e->index; j++)
            {
                if (curve_points[j].curvature > max_curvature)
```

```
                {
                    max_curvature = curve_points[j].curvature;
                    index = i;
                }
            }
        }

        split_points.push_back(fitting_segments[index].e->p);
        split_normals.push_back(fitting_segments[index].e->n);

        //out of deviation
        //split to recursion
        std::vector<FittingElement> fitting_0 = ArcsFitting3D(start_segment_index,
index);
        std::vector<FittingElement> fitting_1 = ArcsFitting3D(index + 1,
end_segment_index);
        fitting_elements_3d = fitting_0;
        for (int i = 0; i < fitting_1.size(); i++)
fitting_elements_3d.push_back(fitting_1[i]);
        std::vector<FittingElement>().swap(fitting_0);
        std::vector<FittingElement>().swap(fitting_1);
        return fitting_elements_3d;
    }
    else
    {
        //during deviation
        fitting_elements_3d.push_back(arc_element);
        return fitting_elements_3d;
    }
}
```

直线段拟合

```
void Fitting::SegmentsFitting3D(int start_curve_index, int end_curve_index,
std::vector<FittingElement> &fitting_segments_3d)
{
    CGAL_3D_Distance_Point_Segment distance =
(CGAL_3D_Distance_Point_Segment)GetProcAddress(hModule,
"CGAL_3D_Distance_Point_Segment");
    //用于计算一个点和一条线段之间的距离
    Vector3d start = curve_points[start_curve_index].p;
    Vector3d end = curve_points[end_curve_index].p;

    //search for the maximal position deviation
```

```
    double max_deviation = -1000.0;
    int index = -1;
    for (int i = start_curve_index+1; i < end_curve_index; i++)
    {
        double d = distance(curve_points[i].p, start, end);
        if (d > max_deviation)
        {
            max_deviation = d;
            index = i;
        }
    }

    //current maximal deviation is bigger than position_deviation
    if (max_deviation > position_deviation)
    {
        //recursion
        std::vector<FittingElement> fitting_0;
        std::vector<FittingElement> fitting_1;
        SegmentsFitting3D(start_curve_index, index, fitting_0);
        SegmentsFitting3D(index, end_curve_index, fitting_1);
        //post-process
        fitting_segments_3d = fitting_0;
        for (int i = 0; i < fitting_1.size(); i++)
            fitting_segments_3d.push_back(fitting_1[i]);
        std::vector<FittingElement>().swap(fitting_0);
        std::vector<FittingElement>().swap(fitting_1);
    }
    else
    {
        fitting_segments_3d.push_back(FittingElement("segment",
&curve_points[start_curve_index], &curve_points[end_curve_index]));
    }
}
```

附录 2 文献英文原文

Robust Circle Detection

Bart Lamiroy, Olivier Gaucher and Laurent Fritz

ABSTRACT

In this paper we present a method of robustly detect circles in a line drawing image. The method is fast, robust and very reliable, and is capable of assessing the quality of its detection. It is based on Random Sample Consensus minimization, and uses techniques that are inspired from object tracking in image sequences.

1. Introduction

Graphical document interpretation or symbol recognition often requires for segmenting or locating symbols, parts of symbols or forms in complex images. Finding circles is one of those issues. The main difficulty with the existing approaches is that they often are of considerable complexity sensible to image quality, line thickness, or rely on a number of user defined parameters or thresholds that make them extremely difficult to apply to generic problems or on heterogeneous document sets. The approach developed in this paper reduces the set of needed parameters to a minimal set of very elementary and visually significant values and can be applied without prior knowledge of the document set, regardless of line widths, connectedness or complexity. It relies on elementary-distance transform skeletonization and circular arc detection. The following section establishes how to determine if a single circle is present, provided we have a rough initial guess of its position, how to robustly detect and locate it using RANSAC. Section 3 then explains how to generalize to detecting and localizing any number of circles, without a priori knowledge of their position. The last two sections conclude by eliminating spurious detections and by establishing the limits of the approach.

2. How To Determine the Presence of a Circle

In this section we address the problem of detecting a circle, given an initial

estimate of its center (x_c, y_c) and its radius σ. This estimate, as we shall see further, can be very approximate. The main goal, in this first stage, is to detect whether or not, a circle is present in the image, near the vicinity of the given center (x_c, y_c).

2.1 General Algorithm

The general approach we develop consists in taking the set P = {pi} of all pixels pi lying on the discrete circle C0defined by (x_c, y_c) and σ. For each of these pixels pi we define the discrete line Δi, starting at (x_c, y_c), and passing through pi. Let qi be the pixel on Δi that is the closest topi and that is black. Let Q0 = {qi}. Q0 therefore is the set of all black pixels closest to the initial estimate C0 in the direction of the circle radius. Now, let C1 be the best fitting circle over Q0, and let us generalize the previous step, such that Qt contains the set of all black pixels closest to the theoretical circle Ct in the direction of the circle ray, and that Ct+1is the best fitting circle over Qt Continuing this iteration until Ct = Ct+1 will yield the best estimate of the circle (if any) closest to the initial C0. In the following sections we are going to detail the different steps of this general approach.

2.2 Using RANSAC and LMedS

Since there is no guarantee that any Qt may effectively contain points that form a circle, it may be extremely hazardous to use global minimization approaches. It is known that these estimators are very sensible to outliers or spurious data that does not conform to the required model. Using these functions would invariably lead to degenerate convergence. RANSAC is much better suited for fitting very noisy data – especially data containing measures that do not belong to the model that is to be estimated – The approach consists in selecting the strict minimum of data points required for estimating an instance of the model (e.g. three points for estimating a circle) and then computing the residual error of the other data points to this model. This is done a number of times, and the final model is the one with the lowest residual error. More formally: let Qt the set of model points. Qt supposedly, and in the worst case, contains a ratio of τ outliers. Let q_n、q'_n and q''_n be three random points

belonging to Qt, and let Cn be the circle defined by and passing through q_n、q'_n and q''_n. Let δ (C, p) be the distance of a point p to a circle C, and let Med τ (S) be the τ -quantile median value of the set S. We then define the residual error of a set of model points Qt to a circle Cn as

$$RsdErr(Q^t, C_n) = Med_r(\{\delta(C_n, p) | p \in Q^t\})$$

RsdErr gives the maximum distance of a set of points to a circle, discarding a proportion of τ outliers. With RANSAC we choose R random subsets of 3 points within Qt, each giving rise to the computation of a circle Cn. For each subset, we compute the corresponding RsdErr (Qt, Cn), thus obtaining

$$C^{t+1} = \underset{C_n, n \in [1 \ldots R]}{\operatorname{argmin}} (RsdErr(Q^t, C_n))$$

2.3 Limitations and Dependency on Initial Estimate

Using only one run for estimating the position of the best circle, corresponding to an initial estimate defined by(xc, yc) and σ may not be a good choice if the estimate is too far off. Figure 1 shows some of these situations. In this example we gave as initial estimate (xc, yc) the center of the image and σ the width of the image. The figure shows in blue the estimation of the best circle, in green, the points correctly corresponding to the circle, and in red, the points closest to the estimated circle. On the left is the initial estimate, and on the right the best fitting circle.

2.4 Data Points and Selection Criteria

In our current approach we do not use the full pixel image for the circle estimation, but we rely on the image skeleton. This is for two main reasons. The first and most essential one concerns the convergence in degenerate situations where the algorithm will naturally have a tendency to converge to very small circles that lie within the thickness of the drawn lines, especially in configurations where lots of intersections occur. This is specially true if the drawing contains filled forms. On the other hand, using skeleton images exclusively, completely eliminates (or at best, heavily distorts) filled shapes. The data used in our algorithm is mainly coming from a skeleton image, in which we added some supplementary treatment in order not to

loose information related to filled shapes. The image first undergoes a thin/thick separation using a simple histogram splitting algorithm. The thick drawing parts (corresponding to the filled shapes) are converted to their contours, and reinjected into the thin part. Only then we proceed to computing the skeleton image.

This approach allows us to correctly identify both filled and non-filled circles, without falling into local minima produced by the infinity of circles that can be drawn within the thick parts of the drawing. This has a drawback, for instance, since we cannot account for the exact positioning of the estimated circle within the line thickness. This is further analyzed in section 6. However, once the best fitting circle (with respect to the skeleton) is found, we add a last selection criterion that will determine if it really corresponds to a circle in the original image. We therefore project it onto the original pixels and retain only those circles for which the number of black pixels exceeds a predetermined ratio (in our case 96%).

3. Filtering Spurious or Multiple Detections

Indeed, numerous circular arcs in the image do not necessarily correspond to full circles in the image, and clutter, intersections and complex symbols may account for multiple circular arcs originating from the same circle in the image. Furthermore, when a circle is split up in several arc segments, it is very likely that various numerical and visual side effects contribute to quite erroneous estimations of the arc center and radius. This inevitably results in poor initial circle estimates that may be quite distinct one from another, although they originally come from the same object. The previously described method will quite elegantly deal with most of the cases: as shown in section 2.3 and Figure 2 poor initial estimates will generally converge to the correct circle. Moreover, our criterion that consists of retaining only those circles that are sufficiently covered by black pixels in the image will efficiently filter out arcs that do not account for full circles. The only remaining issue is the one mentioned in section 2.4: since the RANSAC algorithm only takes into account skeleton data points, and that global verification is only done by substraction of the initial image, any circle correctly covering the image circle will be considered correct. If the

original circle is made of sufficiently thick strokes, the method can converge to several circles for the same image circle. This is, however the only situation in which multiple detections may occur. We currently solve this problem by introducing the MaxRadiusError threshold.

4. Computational Complexity and Performance

Complexity of our method is inherently close to the one presented in with a major difference that the preliminary segmentation and selection process, given by the circular arcs drastically reduces the search space, and that furthermore, the error model is far simpler, without any loss of quality or precision on the one hand (since the underlying minimization goals remain very similar) but with a far higher robustness factor because of the LMeds minimization. Our method, however, because of its initial guess based on arc detecion, reduces the global set of data points, and increases their liability, thus minimizing the number of trials. This selection process does not interfere with global complexitiy since it is done in linear time with respect to the number of points in the image.

5. Conclusions

We have presented in this paper a highly robust method for detecting circles in a line drawing images. It is very fast, extremely robust and reliable, and it is capable of assessing the quality of its detection. It is based on Random Sample Consensus minimization, and uses techniques that are inspired from object tracking in image sequences.

It is, however possible to find configurations where our method fails to find circles. Figure 8 shows some of them. It represents the initial circle hypotheses, showing, for the first symbol that no hypotheses exist for the smaller circles. This is due to the fact that not circular arcs are detected at that level. For the second symbol, the central circle is too cluttered for the circular arcs to be sufficiently robust. This results in the initial circle hypotheses being far off the correct guess. Combined with the characters and strokes within the circle, the RANSAC algorithm gets stuck in a local minimum, failing to detect the correct circle.

附录 3 文献中文译文

鲁棒圆检测

巴特·拉米罗伊，奥利维尔·高彻，劳伦特·弗里茨

摘要

在本文中，我们提出了一种在线条图图像中稳健地检测圆圈的方法。该方法快速、稳健且非常可靠，能够评估其检测质量。它基于随机样本一致性最小化，并使用受图像序列中的对象跟踪启发的技术。

1 引言

图形文档解释或符号识别通常需要在复杂图像中分割或定位符号、符号的一部分或形式。寻找圆圈是这些问题之一。现有方法的主要困难在于它们通常具有相当大的复杂性，对图像质量、线条粗细或依赖于许多用户定义的参数或阈值使它们极难应用于一般性问题或异构文档集。本文开发的方法将所需参数集减少到最小的一组非常基本和视觉上重要的值，并且可以在没有文档集先验知识的情况下应用，无论线宽、连通性或复杂性如何。它依赖于基本的距离变换骨架化和圆弧检测。下一节将介绍如何确定是否存在单个圆圈，前提是我们对其位置有一个粗略的初始猜测，以及如何使用 RANSAC 稳健地检测和定位它。第 3 节然后解释了如何泛化到检测和定位在不知道它们位置的先验知识的情况下绘制任意数量的圆圈。最后两节通过消除虚假检测和建立方法的限制来结束。

2 如何判断圆的存在

在本节中，我们将解决检测圆的问题，给定圆心(x_c, y_c)及其半径σ的初始估计。正如我们将进一步看到的，这个估计可能是非常近似的。第一阶段的主要目标是检测图像中是否存在圆，靠近给定中心(x_c, y_c)。

2.1 通用算法

我们开发的一般方法包括采用位于由(x_c, y_c)和σ定义的离散圆C^0上的所有像素p_i的集合 P=$\{p_i\}$。对于这些像素p_i中的每一个，我们定义离散线Δ_i，从(x_c, y_c)开始，并通过p_i。令q_i是Δ_i上最接近p_i且为黑色的像素。让Q^0=$\{q_i\}$。因此，Q^0是在圆半径方向上最接近初始估计C^0的所有黑色像素的集合。

现在，让C^1成为Q^0上的最佳拟合圆，让我们概括前面的步骤，使得Q^t包含在圆射线方向上最接近理论圆C^t的所有黑色像素的集合，并且C^{t+1}是最佳拟合圆Q^t上继续此迭代直到C^t=C^{t+1}将产生最接近初始C^0的圆的最佳估计。在接下来的部分中，我们将详细介绍这种通用方法的不同步骤。

2.2 使用 RANSAC 和 LMedS 算法

由于无法保证任何Q^t可能有效地包含形成圆的点，因此使用全局最小化方法存在危险。众所周知，这些估计器对不符合所需模型的异常值或虚假数据非常敏感。使用这些函数总是会导致退化收敛。

RANSAC 更适合于处理非常嘈杂的数据——尤其是包含不属于要估计的模型的度量的数据——该方法包括选择估计模型实例所需的严格最小数据点（例如，估计一个圆的三个点）然后计算其他数据点到该模型的残差。重复多次，最终模型是残差最小的模型。

更正式地说：令Q^t为模型点集。据推测，在最坏的情况下，Q^t包含一定比例的τ离群值。设q_n、q_n'和q_n''是属于Q^t的三个随机点，设C_n是由q_n、q_n'和q_n''定义并通过的圆。令$\delta(C,p)$为点p到圆 C 的距离，令Med_r(S)为集合 S 的τ-分位数中值。然后我们定义集合的残差模型点Q^t到一个圆C_n作为

$$RsdErr(Q^t, C_n) = Med_r(\{\delta(C_n, p) | p \in Q^t\})$$

$RsdErr$给出一组点到一个圆的最大距离，丢弃一部分τ离群值。

使用 RANSAC，我们选择Q^t内 3 个点的 R 个随机子集，每个子集都会产生一个圆C_n的计算。对于每个子集，我们计算相应的$RsdErr(Q^t, C_n)$，从而获得

$$C^{t+1} = \underset{C_n, n\in[1\ldots R]}{\operatorname{argmin}} (RsdErr(Q^t, C_n))$$

2.3 对初始估计的限制和依赖

仅使用一次运行来估计最佳圆的位置，对应于由(x_c, y_c)和σ定义的初始估计，如果估计偏离得太远，则可能不是一个好的选择。图 1 显示了其中一些情况。在本例中，我们将图像的中心和σ图像的宽度作为初始估计值(x_c, y_c)。图中显示蓝色是最佳圆的估计，绿色是与圆正确对应的点，红色是最接近估计圆的点。左边是初始估计，右边是最佳拟合圆。可以看出，没有找到圆圈。我们通过重新启动估计过程直到找到的圆稳定来部分解决这个问题。这允许在第一种情况下找到圆，

但在第二个示例中无法找到圆。

2.4 数据点和选择标准

在我们目前的方法中，我们不使用全像素图像进行圆估计，但我们依赖图像骨架。这主要有两个原因。第一个也是最重要的一个问题涉及退化情况下的收敛，在这种情况下，算法自然会倾向于收敛到位于绘制线的粗细范围内的非常小的圆圈，尤其是在出现大量交叉点的配置中。另一方面，只使用骨架图像可以完全消除（或者充其量是严重扭曲）填充形状。我们算法中使用的数据主要来自骨架图像，我们在其中添加了一些补充处理，以免丢失与填充形状相关的信息。图像首先使用简单的直方图分割算法进行薄/厚分离。厚绘图部分（对应于填充的形状）被转换为它们的轮廓，并重新注入到薄部分中。只有这样我们才能继续计算骨架图像。

这种方法使我们能够正确识别填充圆和非填充圆，而不会陷入由可以在绘图的粗部分内绘制的无穷大圆产生的局部最小值。与相比，这有一个缺点，例如，因为我们无法说明估计圆在线宽内的精确定位。这将在第 6 节中进一步分析。但是，一旦找到最佳拟合圆（相对于骨架），我们将添加最后一个选择标准，以确定它是否真的对应于原始图像中的圆。因此，我们将其投影到原始像素上，并仅保留黑色像素数量超过预定比例的那些圆圈。

3. 过滤虚假或多重检测

事实上，图像中的许多圆弧不一定对应图像中的完整圆，杂乱、交叉和复杂的符号可能会导致图像中的同一个圆产生多个圆弧。此外，当一个圆被分成几个圆弧段时，各种数值和视觉副作用很可能导致对圆弧中心和半径的相当错误的估计。这不可避免地导致初始圆圈估计值很差，尽管它们最初来自同一个物体，但它们可能彼此截然不同。前面描述的方法将非常优雅地处理大多数情况：如第 2.3 节和图 2 所示，较差的初始估计通常会收敛到正确的圆。此外，我们的标准包括仅保留那些在图像中被黑色像素充分覆盖的圆圈，将有效地过滤掉那些圆弧考虑不完整的圆圈。唯一剩下的问题是第 2.4 节中提到的问题：由于 RANSAC 算法仅考虑骨架数据点，并且全局验证仅通过减去初始图像来完成，因此任何正确覆盖图像圆的圆都将被认为是正确的。如果原始圆由足够粗的笔画组成，则该方法可以针对同一图像圆收敛到多个圆。然而，这是唯一可能发生多次检测的情况。

我们目前通过引入 MaxRadiusError 阈值来解决这个问题。

4. 计算复杂性和性能

我们的方法的复杂性本质上接近于中提出的方法，主要区别在于由圆弧给出的初步分割和选择过程大大减少了搜索空间，而且错误模型要简单得多，一方面没有任何质量或精度损失（因为潜在的最小化目标仍然非常相似）但由于 LMeds 最小化而具有更高的鲁棒性因子。实际上，需要使用所有方法继续进行试验然而，我们的方法由于其基于电弧检测的初始猜测，减少了全局数据点集，并增加了它们的可能性，从而最大限度地减少了试验次数。这个选择过程不会干扰全局复杂性，因为它是在与图像中的点数相关的线性时间内完成的。

5. 结论

我们在本文中提出了一种用于检测线条画图像中的圆圈的高度鲁棒的方法。它速度非常快、极其稳健和可靠，并且能够评估其检测质量。它基于随机样本共识最小化，并使用受图像序列中的对象跟踪启发的技术。

该方法是一个很难误判圆圈的技巧。然而，在我们的方法无法找到圆的地方找到配置是可能的。图 8 显示了其中的一些。它表示初始圆假说，表明对于第一个符号，较小的圆不存在假说。这是因为在该级别未检测到圆弧。对于第二个符号，中心圆太杂乱，圆弧不够稳健。这导致最初的圈子假设与正确的猜测相去甚远。结合圆圈内的字符和笔画，RANSAC 算法陷入局部最小值，无法检测到正确的圆圈。