

基于 Grasshopper 的参数化设计方法

李钦 2020012872

未央-水木 02

liqin20@mails.tsinghua.edu.cn

摘要

参数化设计是一种基于规则和算法的设计方法, 它可以实现复杂形态的生成和控制, 并提高设计的效率和质量. 本文以 Rhino 和 Grasshopper 为平台, 探讨了参数化设计中的几个关键问题: 曲线打断, 容差合并, 拓扑优化. 首先, 介绍了参数化设计的概念, 特点和发展历程, 以及 Rhino 和 Grasshopper 软件的功能和特点. 其次, 分析了基于树形数据的曲线打断方法, 并给出了相应的 Grasshopper 实现. 然后, 讨论了容差合并的重要性的问题, 并提出了三种容差合并算法: 朴素容差合并算法, 基于近邻搜索的容差合并算法, 基于 k-d Tree 的容差合并算法. 对比了这三种算法的原理, 复杂度和效率, 并给出了相应的 C# 和 Python 源码. 接着, 分析了拓扑优化的重要性, 并介绍了拓扑优化的方法. 最后, 通过一个实际案例, 展示了拓扑优化的实际应用, 并给出了优化流程, 优化结果和优化意义.

关键词: 参数化设计; Rhino; Grasshopper; 曲线打断; 容差合并; 拓扑优化

Abstract

Parametric design is a design method based on rules and algorithms, which can achieve the generation and control of complex forms, and improve the efficiency and quality of design. This paper explores several key issues in parametric design using Rhino and Grasshopper as platforms: curve-breaking, tolerance merging, topological simplification, and repair. Firstly, the concept, characteristics, and development history of parametric design are introduced, as well as the functions and features of Rhino and Grasshopper software. Secondly, the curve-breaking method based on tree-structured data is analyzed, and the corresponding Grasshopper implementation is provided. Then, the importance and problems of tolerance merging are discussed, and three tolerance merging algorithms are proposed: naive tolerance merging algorithm, tolerance merging algorithm based on nearest neighbor search, and tolerance merging algorithm based on k-d tree. The principles, complexity, and efficiency of these three algorithms are compared, and corresponding C# and Python source code are provided. Furthermore, the importance of topological optimization is analyzed, and the methods for topological optimization are introduced. Finally, through a practical case, the practical application of topological optimization is demonstrated, and the optimization process, results, and significance are presented.

Keywords: Parametric Design; Rhino; Grasshopper; Curve Breaking; Tolerance Merging; Topological Optimization

目录

1 引言	4
1.1 参数化设计的概念和特点	4
1.2 参数化设计的发展历程	4
1.3 Rhino 软件介绍	5
1.4 Grasshopper 软件介绍	5
1.4.1 Grasshopper 的功能和特点	5
1.4.2 Grasshopper 在参数化设计中的应用	6
2 曲线打断	7
2.1 曲线打断简介	7
2.2 基于树形数据的曲线打断方法	7
3 容差合并	9
3.1 容差合并的重要性的问题	9
3.2 朴素容差合并算法的原理, 复杂度和效率分析	9
3.3 基于近邻搜索的容差合并算法	10
3.4 基于 k-d Tree 的容差合并算法	12
3.5 容差合并算法的性能比较	13
3.6 点集的排序	15
4 拓扑优化	17
4.1 拓扑优化的重要性	17
4.2 拓扑优化的方法	17
4.3 拓扑优化的实际应用	19
4.3.1 案例介绍	19
4.3.2 优化流程	21
4.3.3 优化结果	21
4.3.4 优化意义	21
5 结论	23
A 附录	24
A.1 C# 源码	24
A.2 Python 源码	40

Listings

1	Listing 1. SplitCurves.cs	7
2	Listing 2. remove_duplicate_points_naive.py	9
3	Listing 3. RemoveDuplicatePoints.cs	10
4	Listing 4. remove_duplicate_points.py	12
5	Listing 5. Simplification.cs	17
6	Listing 6. Graph.cs	24
7	Listing 7. HelloGrasshopperInfo.cs	27
8	Listing 8. RandomGroupCurves.cs	28
9	Listing 9. RandomPoints.cs	31
10	Listing 10. RemoveDuplicatePoints.cs	33
11	Listing 11. Simplification.cs	35
12	Listing 12. SplitCurves.cs	38
13	Listing 13. remove_duplicate_points_naive.py	40
14	Listing 14. remove_duplicate_points.py	41

1 引言

1.1 参数化设计的概念和特点

参数化设计是一种基于参数和变量的设计方法,它通过调整参数的数值或变量的状态来实现设计方案的生成和修改。参数化设计的主要特点是具有灵活性、可重复性和可自动化等特点。

首先,参数化设计具有灵活性,通过设定不同的参数数值,设计师可以生成多种不同的设计方案。这使得设计师能够在较短的时间内尝试不同的设计选项,从而提高设计的多样性和创造性。例如,通过调整建筑立面中的窗户尺寸和布局参数,设计师可以生成多种外观效果和功能布局的设计方案。

其次,参数化设计具有可重复性。通过将设计过程中的关键参数和变量抽象为可调整的参数,设计师可以快速重现之前的设计方案。这对于设计方案的修改、优化和演化等工作非常有价值。当设计需求发生变化时,设计师可以通过修改参数来快速更新设计方案,而不需要从头开始重新设计。这大大提高了设计效率和灵活性。

此外,参数化设计具有可自动化的特点。通过将设计过程中的任务和规则定义为参数化算法,设计师可以将设计任务交给计算机程序来实现。这种自动化的设计过程能够通过计算和优化等算法来寻找最优设计方案,大大提高了设计效率 and 设计质量。例如,Grasshopper 就是一种基于参数化设计的插件,它提供了大量的参数化算法和工具,帮助设计师实现自动化的设计过程。

综上所述,参数化设计具有灵活性、可重复性和可自动化等特点,能够帮助设计师快速生成和修改设计方案。通过将设计过程中的关键参数和变量进行抽象和参数化,设计师可以在较短的时间内生成多样化的设计方案,并通过修改参数实现快速的设计迭代和优化。参数化设计为设计师提供了一种高效、智能和创新的设计方法。

1.2 参数化设计的发展历程

参数化设计是一种利用参数化建模技术进行设计的方法。这种方法通过将设计过程中的各个要素和属性转化为参数,并借助这些参数之间的关系,实现了设计方案的灵活性和高效性。参数化设计的主要优势在于,它可以为设计师提供更大的创造空间和快速迭代的能力,同时也能够更好地满足用户个性化需求。

参数化设计的发展历程可以追溯到上世纪 80 年代。在那个时期,计算机辅助设计 (Computer-Aided Design, 简称 CAD) 开始融入建筑设计领域,设计师们开始意识到参数化建模对于设计的重要性。其中,AutoCAD 是最早在建筑设计中引入参数化思想的软件之一,它的出现让设计师能够通过程序化的方式控制模型的各个要素。

随着计算机技术的不断进步,参数化设计得到了更加广泛的应用。在上世纪 90 年代,虚拟现实技术的发展为参数化设计提供了更多的可能性。同时,建筑信息模型 (Building Information Modeling, 简称 BIM) 的兴起也使得参数化设计在建筑设计中发挥了重要作用。BIM 的核心思想是通过一个统一的数据模型来集成建筑设计和施工过程,而参数化设计可以为 BIM 提供更加灵活和可控的设计手段。

在过去的几十年里,参数化设计的应用范围逐渐扩大,不仅仅局限于建筑设计,还涉及到产品设计,交通规划,城市设计等领域。这得益于诸多参数化设计软件的涌现,其中最为知名的就是 Rhino 和 Grasshopper。

Rhino 是一款三维建模软件,它提供了一个强大的建模环境,可以用于各种设计任务。Rhino 的特点在于其灵活性和可扩展性,设计师可以根据自己的需求进行定制和扩展,实现更加复杂和高效的参数化设计。

Grasshopper 是 Rhino 的一个插件,是一个视觉编程工具,可以实现基于参数化设计的交互式建模。Grasshopper 的出现使得参数化设计的操作变得更加直观和易于掌握。通过 Grasshopper,设计师可以通过连接不同的组件来构建设计流程,并且可以即时预览设计结果,从而更好地理解 and 调整参数。

综上所述, 参数化设计在过去的几十年中取得了长足的进步和发展. 它的优势和应用范围不断扩大, 为设计师提供了更丰富和高效的设计手段. 同时, Rhino 和 Grasshopper 等参数化设计软件的出现也为设计师们带来了更加便捷和灵活的工具. 未来, 参数化设计将继续推动设计领域的创新和发展.

1.3 Rhino 软件介绍

Rhino, 全称 Rhinoceros, 是由美国公司 McNeel 开发的一款三维建模软件. 它被广泛应用于建筑设计, 工业设计, 产品设计, 船舶设计等领域. Rhino 的主要特点是其强大的几何建模功能和可定制性.

Rhino 提供了一个灵活而直观的界面, 使用户可以轻松创建, 编辑和分析三维模型. 它支持各种几何形状, 包括点, 线, 曲线, 曲面, 多边形网格等. 此外, Rhino 还具有丰富的编辑工具, 如平移, 旋转, 缩放, 镜像等, 便于用户对模型进行各种操作和修改.

Rhino 还支持多种文件格式的导入和导出, 如 STEP, IGES, DWG, DXF 等. 这使得用户可以与其他建模软件进行无缝集成, 并与团队成员或客户共享模型数据. 此外, Rhino 还支持流行的 CAD/CAM/CAE 系统, 如 CATIA, SolidWorks, AutoCAD 等.

作为参数化设计的一部分, Rhino 还提供了一些功能来支持参数化建模. 它内置了基本的参数化工具, 如分别对曲线和曲面进行编辑, 创建二维和三维的图形模式, 进行算术和逻辑运算等. 此外, Rhino 还具有高级参数化插件, 如 Grasshopper, 能够提供更强大的参数化建模能力.

在进行参数化设计时, Rhino 的可定制性也是一个重要的优势. 用户可以通过编写脚本和插件来扩展 Rhinoceros 的功能, 以满足特定建模需求. 这种灵活性意味着用户能够根据自己的喜好和需求进行自定义设置, 并将 Rhino 软件调整到最适合自己的工作流程.

总之, Rhino 是一款功能强大, 灵活性高, 易学易用的三维建模软件. 作为参数化设计的工具之一, 它提供了广泛的建模功能和定制性, 帮助设计师在各个领域实现复杂的设计任务.

1.4 Grasshopper 软件介绍

Grasshopper 是一款基于参数化设计的可视化编程软件, 为设计师和建筑师提供了直观强大的设计和分析工具. 本小节将介绍 Grasshopper 的功能和特点, 以及其在参数化设计中的应用.

1.4.1 Grasshopper 的功能和特点

Grasshopper 是一种设计工具, 采用节点和连接线的方式进行参数化设计. 设计师可以通过连接不同的节点, 创建一个流程图, 实现设计的自动化和可视化.

Grasshopper 具有以下重要功能和特点:

- **可视化编程:** Grasshopper 以图形化界面呈现设计过程, 用户可以直观地添加, 删除节点, 并调整参数. 这种可视化编程方式让设计师更好地理解和控制设计过程.
- **参数化建模:** Grasshopper 的核心理念是通过设置和调整参数来控制设计变化. 设计师可以实时观察设计结果的变化, 并根据需要进行优化和调整.
- **强大的几何操作:** Grasshopper 提供丰富的几何操作和算法库, 包括平移, 旋转, 缩放, 融合等操作, 以及数学计算和逻辑运算. 这些功能使得设计师能够灵活处理和生成复杂的几何形态.
- **数据流管理:** Grasshopper 使用连接线将节点连接起来, 形成数据流. 这种方式使得设计过程可追踪和可重复, 设计师可以通过调整节点的顺序和参数, 快速生成不同的设计方案.
- **与 Rhino 的互操作性:** 作为 Rhino 的插件, Grasshopper 与 Rhino 兼容性良好. 设计师可以直接将 Grasshopper 生成的几何形态导入 Rhino 中进行后续的建模, 渲染和分析.

1.4.2 Grasshopper 在参数化设计中的应用

Grasshopper 在参数化设计中有广泛的应用, 包括但不限于以下方面:

- **格栅化设计:** 通过 Grasshopper 的几何操作和算法库, 设计师可以方便地生成各种复杂的格栅化结构. 设计师可以自由调整格栅的参数, 如密度, 大小和形状, 以适应不同的设计需求.
- **结构优化:** Grasshopper 提供了优化算法和插件, 帮助设计师在给定的设计空间内寻找最优解. 设计师可以通过设置目标函数和约束条件, 自动调整设计参数, 以达到结构的最优性能.
- **参数化建模:** 通过 Grasshopper 的参数化建模功能, 设计师可以方便地探索不同的设计方案. 设计师可以通过调整参数, 自动生成设计的多个变体, 比较它们在功能和造型上的差异.

2 曲线打断

2.1 曲线打断简介

曲线打断是参数化设计中常用的一种技术,可以通过在曲线上添加断点将其分割成多段。Grasshopper 是 Rhino 软件中一款强大的可视化编程工具,可用于实现参数化设计中的曲线打断操作。本节将介绍曲线打断的原理及其在参数化设计中的应用。

在参数化设计中,曲线打断是一个常见的操作,可以让设计师对曲线进行细粒度的控制。通过曲线打断,设计师可以将曲线划分成多个片段,并对其进行个别处理。这种操作可以用于构建复杂的几何形状,进行曲线上特定区域的编辑,以及生成符合设计要求的曲线样式。

通过 Grasshopper 的强大功能,曲线打断可以快速实现。设计师可以使用 Grasshopper 的曲线工具创建自定义的曲线,然后使用打断工具将其分割成多个片段。打断工具允许设计师在曲线上添加断点,并根据需要调整断点的位置和数量。设计师还可以使用 Grasshopper 的其他组件对打断后的曲线片段进行进一步的操作和修改,以满足设计的要求。

曲线打断的应用广泛存在于各个设计领域。在建筑设计中,曲线打断可以用于生成具有动态感的曲线体量,提升建筑的空间层次感。在产品设计中,曲线打断可以用于创建独特的产品造型,增强产品的美观性和功能性。在景观设计中,曲线打断可以用于创造流畅的路径线路,使景观布局更加灵活多样。

2.2 基于树形数据的曲线打断方法

曲线打断方法的有效实现对于参数化软件的应用具有重要意义。尽管在 Grasshopper 中提供了曲线打断工具,但该工具存在一些局限性,例如无法处理树形结构存储的曲线以及丢失曲线的属性信息。为了解决这些问题,我们在本研究中提出了一个新的曲线打断插件。

Listing 1. SplitCurves.cs

```
45 protected override void SolveInstance(IGH_DataAccess DA)
46 {
47     DA.GetDataTree(name: "Curves", tree: out GH_Structure<GH_Curve> curves);
48     GH_Structure<GH_Curve> result = new GH_Structure<GH_Curve>();
49     foreach (GH_Path path in curves.Paths)
50     {
51         for (int i = 0; i < curves[path].Count; ++i)
52         {
53             Curve curve = curves[path][i].Value;
54             List<double> parameters = new List<double>();
55             foreach (GH_Curve gh_cutter in curves.AllData(skipNulls: true))
56             {
57                 Curve cutter = gh_cutter.Value;
58                 if (cutter == curve)
59                     continue;
60                 CurveIntersections intersections = Intersection.CurveCurve(
61                     curve,
62                     cutter,
```

```
63         tolerance: 0.0,  
64         overlapTolerance: 0.0  
65     );  
66     parameters.AddRange(  
67         intersections.Select(intersection => intersection.ParameterA)  
68     );  
69     }  
70     parameters.Sort();  
71     parameters = parameters  
72         .Prepend(curve.Domain.Min)  
73         .Append(curve.Domain.Max)  
74         .ToList();  
75     List<GH_Curve> segments = new List<GH_Curve>();  
76     for (int j = 0; j + 1 < parameters.Count; ++j)  
77     {  
78         double t0 = parameters[j];  
79         double t1 = parameters[j + 1];  
80         segments.Add(new GH_Curve(curve.Trim(t0, t1)));  
81     }  
82     result.AppendRange(data: segments, path: path.AppendElement(i));  
83 }  
84 }  
85 DA.SetDataTree(paramIndex: 0, tree: result);  
86 }
```

我们的曲线打断插件的核心代码如 listing 1 所示, 完整代码参见 listing 12. 首先, 我们通过输入参数 `Curves` 获取一个曲线树结构, 该结构以路径为单位存储了一组曲线. 然后, 我们遍历树结构中的所有路径, 并逐个处理其中的曲线.

对于每个曲线, 我们首先创建一个空的曲线结果集合, 用于存储被分割后的曲线段. 然后, 我们针对当前曲线和其他曲线进行相交检测. 如果两条曲线相同, 我们将继续处理下一条曲线. 否则, 我们使用 `Intersection.CurveCurve` 方法计算曲线相交的交点, 并将其中一个曲线的参数值加入到参数集合中.

接下来, 我们对参数集合进行排序, 并将当前曲线的起始和结束参数值加入到参数集合中. 然后, 我们以参数集合中相邻参数值的位置为界, 将当前曲线分割成多个曲线段, 并将这些曲线段作为新的曲线对象加入到曲线结果集合中.

最后, 我们将曲线结果集合添加到树结构中的对应路径中, 完成所有曲线的打断操作. 最终, 通过输出参数可以获取到完整的打断后的曲线树结构.

我们的曲线打断插件能够有效处理树形结构存储的曲线, 并且能够保留曲线的属性信息. 这为参数化软件的用户提供了更灵活和便捷的曲线操作方式, 进一步提升了其应用领域的实际价值.

本研究的实现为曲线打断方法在 Rhino 和 Grasshopper 中的应用提供了一种可靠而高效的解决方案. 通过借鉴和优化现有工具的不足之处, 我们为参数化软件的开发和使用者提供了一种优化的曲线打断功能, 从而进一步推进了参数化设计的应用和发展.

3 容差合并

3.1 容差合并的重要性和问题

容差合并是参数化设计中一个重要的步骤,旨在解决由于建模过程中引入的误差和不精确性所导致的重复点的问题。这一步骤通常发生在将参数化模型转化为实际产品的过程中。

容差合并的重要性在于以下方面:

- **精确性:** 容差合并能够确保设计模型的准确性和精确性。在参数化设计中,由于建模过程中的操作误差和计算机近似等因素,可能会导致重复点的出现。通过进行容差合并,可以消除这些重复点,从而提高模型的精确性和准确性。
- **可视化:** 容差合并还有助于改善设计模型的可视化效果。重复点会在可视化过程中产生杂乱的效果,使设计师难以准确地观察和理解模型的细节。通过容差合并,可以使设计模型更加清晰,整洁,为设计师提供更好的可视化效果。
- **优化操作:** 容差合并可以简化模型编辑和操作的过程。重复点存在时,设计师需要额外的操作来清理和调整模型。而通过容差合并,设计师可以更方便地进行模型编辑和操作,提高工作效率。

然而,容差合并也存在一些问题和挑战:

- **容差选择:** 选择合适的容差是容差合并过程中的一个关键问题。过小的容差可能会导致重要的模型特征丢失,而过大的容差可能会导致不必要的模糊和模型失真。设计师需要根据具体的设计要求和实际情况来选择合适的容差。
- **数据处理:** 容差合并需要对大量的数据进行处理和计算。当模型复杂度增加时,计算量和处理时间也会增加。设计师需要采用高效的算法和合适的计算资源来处理和优化这些数据,以提高容差合并的效率和效果。
- **误删问题:** 容差合并可能会误删设计模型中的一些细节和特征。在合并过程中,一些本应存在的点可能会被错误地删除。为了避免这种问题,设计师需要进行仔细的检查和验证,确保容差合并不会对设计模型产生负面影响。

综上所述,容差合并参数化设计中起着重要的作用。它可以提高模型的精确性和可视化效果,优化操作流程,但也需要面对容差选择,数据处理和误删等问题。

3.2 朴素容差合并算法的原理,复杂度和效率分析

本节将介绍一种朴素的容差合并算法,该算法能够从一个给定的点列表中去掉相似的点。该算法的目的是通过合并距离小于给定阈值的点来减少点的数量。

Listing 2. remove_duplicate_points_naive.py

```
19 def point_less(u, v):
20     u_coordinates = rs.PointCoordinates(u)
21     v_coordinates = rs.PointCoordinates(v)
22     return u_coordinates < v_coordinates
23
24
25 removed = set()
26 for u, v in itertools.combinations(P, 2):
```

```

27     if u in removed or v in removed:
28         continue
29     if not rs.PointCompare(u, v, tolerance=t):
30         continue
31     removed.add(v if point_less(u, v) else u)
32 Q = [u for u in P if u not in removed]

```

算法使用了 Rhino 和 Grasshopper 中的 `rhinoscriptsyntax` 库作为辅助工具。通过使用此库中的函数可以轻松获取和操作点的坐标。

算法的输入包括一个点列表 `P` 和一个阈值 `t`，表示距离小于 `t` 的点将会被合并。算法的输出是一个不包含重复点的新的点列表 `Q`。

首先，我们初始化一个集合 `removed` 用于存储已经被移除的点。接下来，我们使用 `itertools` 库中的 `combinations` 函数对点列表 `P` 中的每一对点进行组合。通过对每一对点进行比较，我们可以判断它们是否相似。

在比较两个点的相似性时，首先检查这两个点是否已经被移除。如果其中一个点已经被移除，则可以跳过此对点的比较。

如果两个点的距离小于给定的阈值 `t`，则它们被认为是相似的。在这种情况下，我们将选择其中一个点添加到 `removed` 集合中。为了保持结果的一致性，我们根据一个自定义的 `point_less` 函数来选择保留哪个点。该函数将比较两个点的坐标，并返回一个布尔值表示哪个点的坐标较小。根据返回的结果，我们添加较小的点到 `removed` 集合中。

最后，我们根据 `removed` 集合将点列表 `P` 筛选为新的点列表 `Q`。具体而言，我们遍历列表 `P` 中的每一个点，并判断每个点是否在 `removed` 集合中。如果点不在 `removed` 集合中，则将其添加到新的点列表 `Q` 中。

该算法的时间复杂度为 $\Theta(n^2)$ ，其中 n 是输入点列表的长度。因为算法使用了两个嵌套的循环来比较每一对点，所以算法的效率在处理大型输入时可能较低。

然而，该算法的优点在于其简单性和易于实现，特别是在处理相对较小的点列表时是一种可行的选择。此外，该算法不需要对点列表进行排序，因此可以在不影响结果的前提下保持点的相对顺序。

如果对于大型点列表而言，效率是非常重要的需求，那么更高效的算法可以使用空间来进行换取，例如构建临近点的索引结构，以减少比较操作的时间复杂度。但是这种算法可能会更加复杂，并且需要额外的空间来存储索引结构。

因此，选择使用哪种算法取决于具体的需求和时间 / 空间权衡。对于一些简单的应用场景，朴素算法足够实现要求；对于更复杂的情况，可以进一步研究和实现更高效的算法。

3.3 基于近邻搜索的容差合并算法

在本节中，我们介绍一种基于近邻搜索的容差合并算法，该算法可以有效地处理大量的点集，以合并距离小于给定容差的重复点。该算法利用了 Rhino 和 Grasshopper 的空间数据结构和查询功能，以提高合并效率和准确性。

Listing 3. RemoveDuplicatePoints.cs

```

49 protected override void SolveInstance(IGH_DataAccess DA)
50 {

```

```

51 List<Point3d> points = new List<Point3d>();
52 double tolerance = 0.01;
53 DA.GetDataList("Points", points);
54 DA.GetData("Tolerance", ref tolerance);
55 points.Sort((Point3d a, Point3d b) => a.CompareTo(b));
56 HashSet<Point3d> point_set = new HashSet<Point3d>(points);
57 Node3d<Point3d> tree = new Node3d<Point3d>(
58     converter: (Point3d pt, out double x, out double y, out double z) =>
59     {
60         x = y = z = 0.0;
61         TreeDelegates.Point3dCoordinates(pt, ref x, ref y, ref z);
62     },
63     region: new BoundingBox(points)
64 );
65 tree.AddRange(points);
66 for (int i = 0; i < points.Count; i++)
67 {
68     if (!point_set.Contains(points[i]))
69         continue;
70     while (true)
71     {
72         List<Index3d<Point3d>> neighbors = tree.NearestItems(
73             locus: points[i],
74             groupSize: 2,
75             minimumDistance: 0.0,
76             maximumDistance: tolerance
77         );
78         if (neighbors.Count < 2)
79             break;
80         Index3d<Point3d> neighbor = neighbors[1];
81         if (points[i].DistanceTo(neighbor.Item) > tolerance)
82             break;
83         point_set.Remove(neighbor.Item);
84         tree.Remove(neighbor);
85     }
86 }
87 DA.SetDataList("UniquePoints", point_set);
88 }

```

该算法的输入是一个点集和一个容差值, 输出是一个去除重复点后的点集. 该算法的主要步骤如下:

1. 将点集按照 `x`, `y`, `z` 坐标进行排序, 以便后续的查找和删除操作.
2. 创建一个空的 `HashSet`, 用于存储最终的结果.

3. 创建一个空间划分树 `Node3d`, 用于存储和查询点集中的点. 该树使用 Rhino 的 `BoundingBox` 作为划分区域, 使用 `TreeDelegates.Point3dCoordinates` 作为坐标转换器.
4. 将点集中的所有点添加到 `HashSet` 和空间划分树中.
5. 遍历点集中的每个点, 如果该点已经被从 `HashSet` 中移除, 则跳过该点; 否则, 执行以下操作:
 - (a) 使用空间划分树的 `NearestItems` 方法, 查找距离该点在容差范围内的最近的两个点 (包括该点本身).
 - (b) 如果只找到一个点 (即该点本身), 则说明该点没有重复点, 继续下一个点的遍历.
 - (c) 如果找到两个或更多的点, 则说明该点有重复点, 取第二个最近的点作为重复点, 并从 `HashSet` 和空间划分树中移除该重复点. 然后重复这一步骤, 直到找不到更多的重复点为止.
6. 将 `HashSet` 中剩余的点作为输出返回.

该算法的时间复杂度主要取决于两个方面: 排序和查找. 排序的时间复杂度为 $\Theta(n \log n)$, 其中 n 是点集的大小. 查找的时间复杂度为 $\Theta(k \log n)$, 其中 k 是每个点的平均重复数. 因此, 总的时间复杂度为 $\Theta(kn \log n)$. 相比于朴素算法, 该算法利用了空间划分树来加速查找过程, 避免了对整个点集的遍历, 从而提高了效率.

该算法的空间复杂度主要取决于 `HashSet` 和空间划分树所占用的空间. `HashSet` 的空间复杂度为 $\Theta(n)$, 空间划分树的空间复杂度为 $\Theta(n)$. 因此, 总的空间复杂度为 $\Theta(n)$. 相比于朴素算法, 该算法没有使用额外的数组或列表来存储临时结果, 从而节省了空间.

该算法的优势在于可以处理大量的点集, 并且可以灵活地调整容差值. 该算法也可以很容易地扩展到其他类型的几何对象, 如线段, 曲线, 面等. 该算法的缺陷在于需要对输入进行排序, 这可能会改变原始数据的顺序; 另外, 该算法也依赖于 Rhino 和 Grasshopper 的内置功能, 如果在其他平台上实现, 则可能需要自行编写相应的数据结构和查询方法.

3.4 基于 k-d Tree 的容差合并算法

在本节中, 我们介绍一种基于 k-d 树的容差合并算法, 该算法可以有效地处理大量的点云数据, 并在给定的容差范围内合并重复的点. 该算法的主要思想是利用 k-d 树的空间划分和近邻搜索的特性, 来快速找出距离小于容差的点, 并将它们替换为最小的点 ID. 该算法的核心代码如 listing 4 所示, 完整代码参见 listing 14:

Listing 4. `remove_duplicate_points.py`

```

11 kdtree: KDTree = KDTree(data=points)
12 results: set[int] = set(range(points.shape[0]))
13 for i in np.lexsort(keys=points[:, :-1].transpose()):
14     if i not in results:
15         continue
16     duplicates: set[int] = (
17         set(kdtree.query_ball_point(x=points[i], r=tolerance)) & results
18     )
19     min_id: int = min(duplicates, key=lambda x: tuple(points[x]))
20     results -= set(duplicates)
21     results.add(min_id)

```

该算法的时间复杂度为 $\Theta(n \log n + kn)$, 其中 n 是点的数量, k 是平均每个点的近邻数量. 该算法的空间复杂度为 $\Theta(n)$, 因为它只需要存储点云数据和 k-d 树结构. 相比于朴素的容差合并算法, 该算法有以下优势:

- 该算法可以处理任意维度的点云数据, 而朴素算法只能处理二维或三维的情况.
- 该算法可以利用 k-d 树的高效近邻搜索, 避免了对所有点进行两两比较, 从而降低了时间复杂度.
- 该算法可以通过对点云数据进行排序, 保证了合并后的点 ID 是最小的, 从而避免了对点 ID 的重排.

该算法也有一些局限性, 例如:

- 该算法需要构建 k-d 树结构, 会增加一定的空间开销和预处理时间.
- 该算法对于容差较大或点云密度不均匀的情况, 可能会产生过度合并或漏合并的问题.
- 该算法对于点云数据中存在噪声或异常值的情况, 可能会影响合并结果的质量.

因此, 在使用该算法时, 需要根据具体的应用场景和数据特征, 选择合适的容差参数和预处理方法, 以达到最佳的效果.

3.5 容差合并算法的性能比较

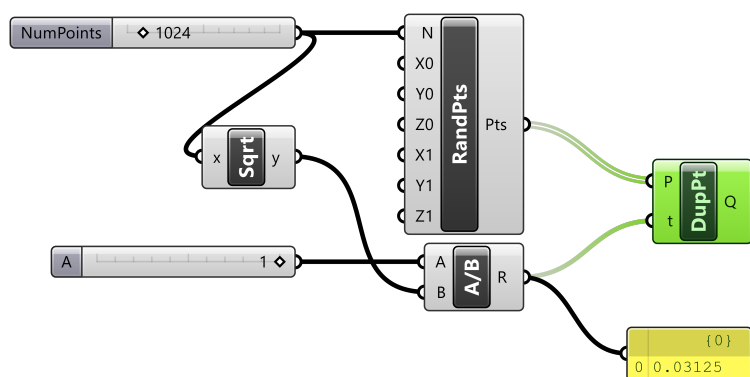


图 1: 容差合并测试流程

点的容差合并算法是一种在参数化建模中常用的技术, 它可以将距离小于一定阈值的点合并为一个点, 从而简化模型的复杂度和提高运算效率. 本文实现了三种点的容差合并算法, 分别是 **Naive**, **Tree3d** 和 **k-d Tree**, 并在不同规模的点集上进行了测试, 测试方法如 fig. 1 所示, 比较了它们的运行时间和合并效果, 测试结果如 figs. 2 and 3 and table 1 所示. 下面是测试结果的分析:

- **Naive** 算法是一种最简单的算法, 它遍历所有的点对, 计算它们之间的距离, 如果小于给定的容差, 则将它们合并为一个点. 这种算法的时间复杂度是 $\Theta(n^2)$, 其中 n 是点的个数. 从表格中可以看出, 当点的个数增加时, **Naive** 算法的运行时间呈指数增长, 当点的个数超过 2048 时, 运行时间就超过了 2s, 当点的个数达到 8192 时, 运行时间就超过了 10s. 这说明 **Naive** 算法在处理大规模的点集时效率非常低下, 不适合用于实际应用.
- **Tree3d** 算法是一种基于空间划分的算法, 它利用了 C# 语言提供的三维树结构, 将点集划分为若干个立方体区域, 然后对每个区域内的点进行容差合并. 这种算法的时间复杂度是 $\Theta(kn \log n)$, 其中 n 是点的个数, k 是平均每个点的近邻数量. 从表格中可以看出, **Tree3d** 算法在处理不同规模的点集时都有较好的性能, 运行时间随着点的个数增加而增加, 但增长速度较慢. 当点的个数达到 32768 时, 运行时间仍然在 6.5s 以内. 这说明 **Tree3d** 算法在处理大规模的点集时效率较高, 适合用于实际应用.

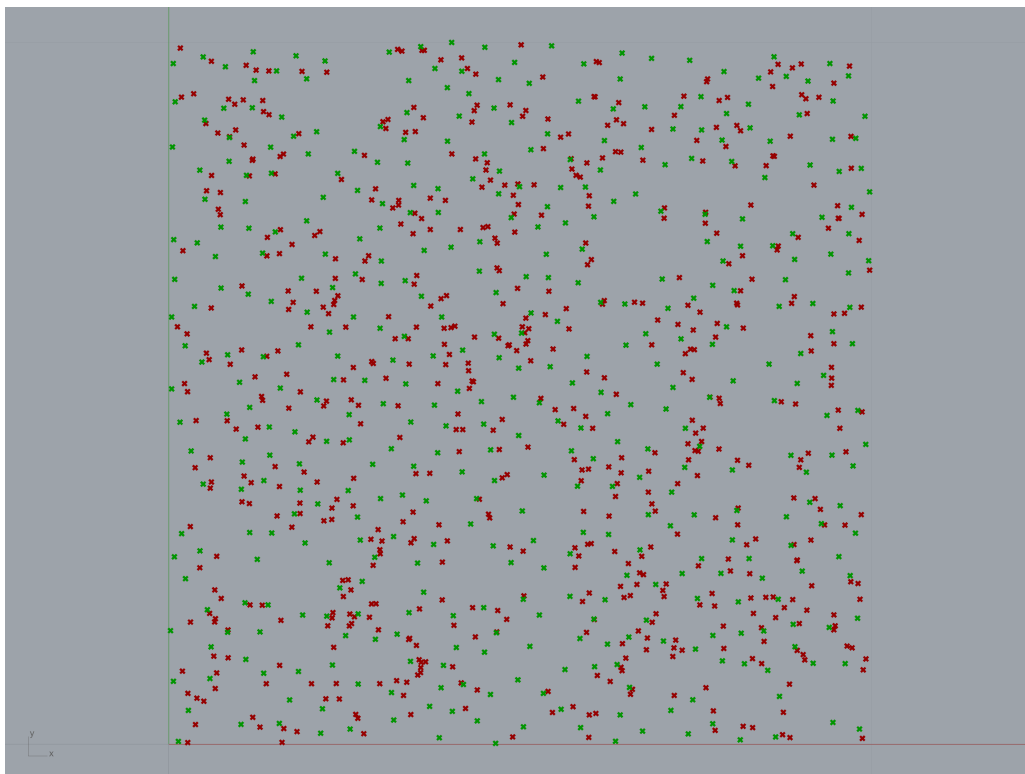


图 2: 容差合并结果 (保留, 删除)

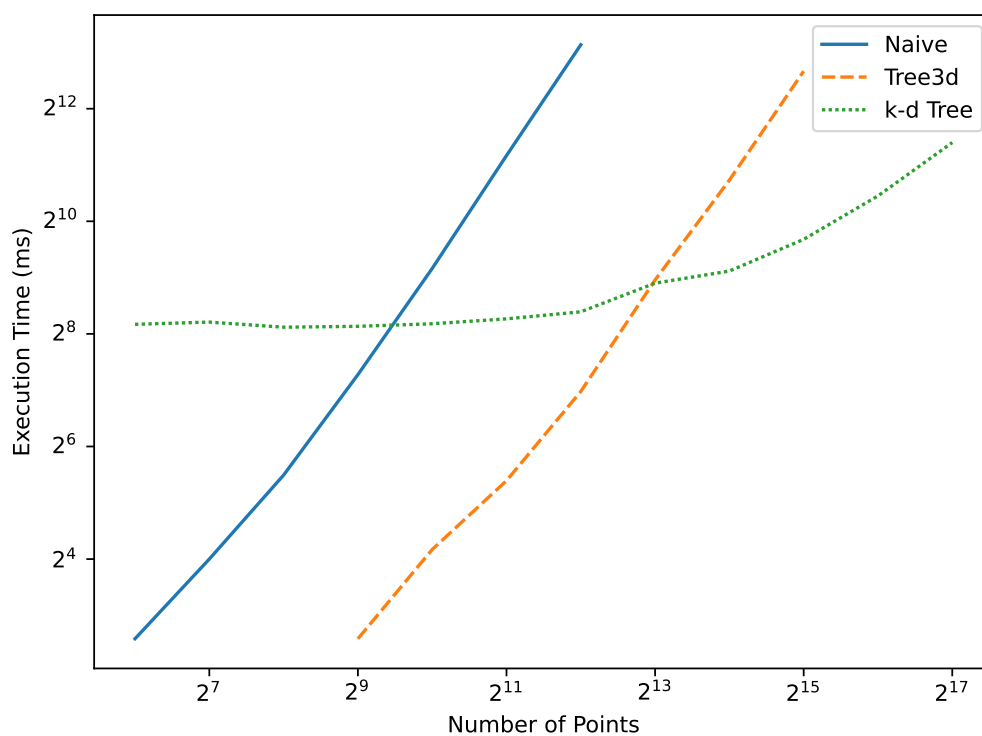


图 3: 容差合并算法的性能测试结果

表 1: 容差合并算法的性能测试结果

Num Points	Tolerance	Naive	Tree3d	k-d Tree
64	0.125 000	6 ms		288 ms
128	0.088 388	16 ms		296 ms
256	0.062 500	45 ms		278 ms
512	0.044 194	155 ms	6 ms	281 ms
1024	0.031 250	573 ms	18 ms	290 ms
2048	0.022 097	2.3 s	42 ms	308 ms
4096	0.015 625	9.0 s	126 ms	336 ms
8192	0.011 049		497 ms	478 ms
16 384	0.007 813		1.7 s	555 ms
32 768	0.005 524		6.5 s	820 ms
65 536	0.003 906			1.4 s
131 072	0.002 762			2.7 s

- **k-d Tree** 算法是一种基于递归划分的算法, 它利用了 **scipy** 提供的 k-d 树结构, 将点集按照每个维度依次划分为两个子集, 然后对每个子集内的点进行容差合并. 这种算法的时间复杂度也是 $\Theta(n \log n + kn)$, 其中 n 是点的个数, k 是平均每个点的近邻数量. 从表格中可以看出, **k-d Tree** 算法在处理不同规模的点集时都有较好的性能, 运行时间随着点的个数增加而增加, 但增长速度较慢. 当点的个数达到 131 072 时, 运行时间仍然在 2.7 s 以内. 这说明 **k-d Tree** 算法在处理大规模的点集时效率较高, 适用于实际应用.

综上所述, 我们可以得出以下结论:

- 在三种算法中, **Naive** 算法是最简单但也最低效的算法, 在处理大规模的点集时不可取.
- 在三种算法中, **Tree3d** 和 **k-d Tree** 算法都是基于空间划分或递归划分的高效算法, 在处理大规模的点集时都有较好的性能.
- 由于调用了外部 Python, 存在较大的通信和启动开销, 因此 **k-d Tree** 算法相比于 **Tree3d** 算法有一定程度上的性能损失.

3.6 点集的排序

点的容差合并算法的一个关键步骤是对点集进行排序, 以保证合并结果的稳定性和正确性. 在本节中, 我们将介绍我们采用的排序算法, 并证明它的必要性和有效性.

我们首先定义点集上的全序关系为 \leq , 它满足以下性质:

- **反对称性**: 对于任意点 v_1, v_2 , 如果 $v_1 \leq v_2$ 且 $v_2 \leq v_1$, 则 $v_1 = v_2$.
- **传递性**: 对于任意点 v_1, v_2, v_3 , 如果 $v_1 \leq v_2$ 且 $v_2 \leq v_3$, 则 $v_1 \leq v_3$.
- **完全性**: 对于任意点 v_1, v_2 , 要么 $v_1 \leq v_2$, 要么 $v_2 \leq v_1$.

我们可以根据点的坐标来定义这样一个算符. 例如, 如果点的维数为 d , 我们可以按照字典序来比较点的坐标. 即, 对于两个点 $v_1 = (x_{11}, x_{12}, \dots, x_{1d})$ 和 $v_2 = (x_{21}, x_{22}, \dots, x_{2d})$, 我们有:

$$v_1 < v_2 \iff \exists i \in \{1, 2, \dots, d\}, \forall j \in \{1, 2, \dots, i-1\}, x_{1j} = x_{2j} \wedge x_{1i} < x_{2i}$$

这样定义的算符满足上述性质, 并且可以在 $O(d)$ 的时间内比较两个点.

其次, 我们介绍如何删除重复点. 给定一个点集 P 和一个容差参数 t , 我们的目标是找到一个子集 Q , 使得 Q 中的任意两个点之间的距离都大于或等于 t , 且 Q 包含了 P 中所有不可删除的点. 我们称一个点是不可删除的, 如果它在容差范围内没有比它更小的点. 我们的删除方法如下:

1. 对点集 P 按照算符 $<$ 进行排序, 得到一个序列 $\{v_1, v_2, \dots, v_n\}$.
2. 从左到右遍历序列, 对于每个点 v_i , 检查它是否在容差范围内有比它更小的点. 如果有, 则将其标记为可删除的; 如果没有, 则将其加入到子集 Q 中.
3. 返回子集 Q 作为最终的合并结果.

接下来, 我们举例说明合理排序的必要性. 假设我们有三个点 $v_1 = (0, 0)$, $v_2 = (0.5, 0)$ 和 $v_3 = (1, 0)$, 且容差参数为 $t = 0.6$. 如果我们按照从小到大的顺序处理这些点, 那么我们会得到子集 $\{v_1, v_3\}$; 如果我们按照从大到小的顺序处理这些点, 那么我们会得到子集 $\{v_1\}$. 可见, 处理顺序会影响合并结果的稳定性.

最后, 我们证明我们的排序方法是正确的. 即对于任意一个点集 P 和一个容差参数 t , 我们的方法能够找到一个包含所有不可删除点的子集 Q . 为了证明这一点, 我们需要以下两个引理:

Lemma 3.1

如果一个点是不可删除的, 那么它一定是最小的点或者它左边没有在容差范围内的点.

Lemma 3.2

如果一个点是可删除的, 那么它一定有一个比它更小且在容差范围内的不可删除点.

Lemma 3.1 的证明很简单: 如果一个点既不是最小的点, 又有左边在容差范围内的点, 那么它就可以被这个左边的点替代, 因此是可删除的.

Lemma 3.2 的证明如下: 设一个可删除点为 v_i , 那么它一定有一个比它更小且在容差范围内的点 v_j , 其中 $j < i$. 如果 v_j 是不可删除的, 那么引理成立. 如果 v_j 是可删除的, 那么根据 lemma 3.1, 它一定有一个比它更小且在容差范围内的点 v_k , 其中 $k < j$. 我们可以重复这个过程, 直到找到一个不可删除的点为止. 这个过程一定会终止, 因为序列是有限的, 而且最小的点一定是不可删除的.

有了这两个引理, 我们就可以证明我们的方法是正确的. 设我们的方法得到的子集为 Q , 我们需要证明 Q 包含了所有不可删除点. 对于任意一个不可删除点 v_i , 根据 lemma 3.1, 它一定是最小的点或者它左边没有在容差范围内的点. 如果它是最小的点, 那么它一定会被我们的方法加入到 Q 中; 如果它左边没有在容差范围内的点, 那么当我们遍历到它时, 我们也会将它加入到 Q 中. 因此, 对于任意一个不可删除点, 我们都能保证它在 Q 中.

综上所述, 我们证明了我们的排序方法是正确的, 并且能够保证合并结果的稳定性.

4 拓扑优化

4.1 拓扑优化的重要性

拓扑优化是参数化设计中的一个重要的步骤，它可以提高数据的质量和效率，并为后续的设计和分析过程提供更好的基础。拓扑优化的目的是将复杂或混乱的曲线和点转化为简单或规整的曲线和点，并保留其原有的形状和拓扑关系。

拓扑优化的重要性可以从以下几个方面体现：

- 拓扑优化可以减少数据的冗余和噪声，从而节省存储空间和计算时间。例如，如果一个曲线由多个重叠或相近的线段组成，那么可以将这些线段合并为一个线段，从而减少数据量和计算量。
- 拓扑优化可以消除数据中的错误或异常，从而提高数据的准确性和可靠性。例如，如果一个曲线存在自交或环路，那么可以将这些部分删除或分割，从而避免造成混淆或错误。
- 拓扑优化可以优化数据的结构和形式，从而提高数据的可读性和可操作性。例如，如果一个曲线由多个不连续或不平滑的线段组成，那么可以将这些线段连接或平滑，从而使曲线更加连续或平滑。
- 拓扑优化可以为后续的设计和分析过程提供更好的基础。例如，如果需要将模型导入有限元软件进行计算，那么需要对模型进行网格划分。如果模型中存在混乱或重复的曲线，那么可能导致无法正确生成网格划分，或者生成低质量或不合理的网格划分。而如果对模型进行了拓扑优化，那么可以使模型更加符合网格划分的要求，并生成高质量或合理的网格划分。

因此，拓扑优化是参数化设计中不可或缺的一个环节，它可以极大地提升参数化设计的效果和效率。

4.2 拓扑优化的方法

在参数化设计中，往往需要处理大量的曲线和点，这些曲线和点可能来自于不同的数据源，例如 Rhino 的模型，Grasshopper 的组件，或者其他的软件或设备。这些曲线和点可能存在一些问题，例如重复，交叉，环路，孤立等，这些问题会影响到后续的设计和分析过程。因此，需要对这些曲线和点进行拓扑优化，以提高数据的质量和效率。

本文提出了一种拓扑优化的方法，该方法基于图论的概念，将曲线和点转化为图的顶点和边，然后对图进行一系列的操作，以达到优化的目的。该方法使用了 Rhino 和 Grasshopper 作为平台，并利用了 C# 语言编写了一个自定义的 Grasshopper 组件。该组件的输入参数包括：

- **Curves**：一个或多个需要处理的曲线
- **Tolerance1**：第一次简化时使用的距离容差
- **Tolerance2**：第二次简化时使用的距离容差

该组件的输出参数包括：

- **UniqueCurves**：经过拓扑优化后得到的唯一曲线
- **UniquePoints**：经过拓扑优化后得到的唯一点

该组件的核心代码如 listing 5，完整代码参见 listings 6 and 11。

Listing 5. Simplification.cs

```
62 protected override void SolveInstance(IGH_DataAccess DA)
63 {
64     List<Curve> curves = new List<Curve>();
```

```

65     double tolerance_1 = 0.0;
66     double tolerance_2 = 0.0;
67     DA.GetDataList(name: "Curves", list: curves);
68     DA.GetData(name: "Tolerance1", destination: ref tolerance_1);
69     DA.GetData(name: "Tolerance2", destination: ref tolerance_2);
70     Graph<Point3d> graph = new Graph<Point3d>();
71     foreach (Curve curve in curves)
72     {
73         foreach (Curve segment in curve.DuplicateSegments())
74         {
75             Point3d start = segment.PointAtStart;
76             Point3d end = segment.PointAtEnd;
77             graph.AddVertex(start);
78             graph.AddVertex(end);
79             graph.AddEdge(start, end);
80         }
81     }
82     Simplify(ref graph, tolerance_1);
83     Simplify(ref graph, tolerance_2);
84     graph.RemoveDegree2Vertices();
85     List<LineCurve> unique_curves = new List<LineCurve>();
86     foreach (Edge<Point3d> e in graph.edges)
87         unique_curves.Add(new LineCurve(from: e.from.data, to: e.to.data));
88     DA.SetDataList(paramName: "UniqueCurves", data: unique_curves);
89     List<Point3d> unique_points = new List<Point3d>();
90     foreach (Vertex<Point3d> v in graph.vertices)
91         unique_points.Add(v.data);
92     DA.SetDataList(paramName: "UniquePoints", data: unique_points);
93 }
94
95 internal void Simplify(ref Graph<Point3d> graph, double tolerance = 0.01)
96 {
97     List<Vertex<Point3d>> vertices = new List<Vertex<Point3d>>(graph.vertices);
98     vertices.Sort(
99         (Vertex<Point3d> u, Vertex<Point3d> v) =>
100     {
101         if (u.adjacencies.Count == v.adjacencies.Count)
102             return u.data.CompareTo(v.data);
103         else
104             return v.adjacencies.Count.CompareTo(u.adjacencies.Count);
105     }
106 );

```

```

107     foreach (Vertex<Point3d> u in vertices)
108     {
109         if (!graph.vertices.Contains(u))
110             continue;
111         List<Vertex<Point3d>> to_remove = new List<Vertex<Point3d>>();
112         foreach (Vertex<Point3d> v in graph.vertices)
113         {
114             if (u == v)
115                 continue;
116             if (u.data.DistanceTo(v.data) < tolerance)
117                 to_remove.Add(v);
118         }
119         graph.ContractVertices(root: u, vertices: to_remove);
120     }
121     graph.RemoveLoops();
122     graph.RemoveIsolatedVertices();
123 }

```

该算法主要包括以下几个步骤:

1. 将输入的曲线分割成若干段线段, 并将每个线段的起点和终点作为图的顶点, 每个线段作为图的边.
2. 对图进行第一次简化, 即将距离小于 `Tolerance1` 的顶点合并为一个顶点, 并删除相应的边.
3. 对图进行第二次简化, 即将距离小于 `Tolerance2` 的顶点合并为一个顶点, 并删除相应的边.
4. 对图进行清理, 即删除度为 2 的顶点及其相邻的边, 删除自环边, 删除孤立顶点.
5. 将图中剩余的顶点和边转换为输出参数 `UniquePoints` 和 `UniqueCurves`.

该算法具有以下特点:

- 可以处理任意形状和数量的曲线
- 可以根据不同的容差值进行多次简化
- 可以有效地消除重复, 交叉, 环路, 孤立等问题
- 可以保留曲线之间的拓扑关系
- 可以在 Rhino 和 Grasshopper 中方便地调用和使用

4.3 拓扑优化的实际应用

在本节中, 我们将展示拓扑优化方法在实际案例中的应用效果. 我们选择了一个复杂的曲线模型, 并使用 Rhino 和 Grasshopper 实现了拓扑优化的流程. 我们将对比优化前后的模型, 并分析优化的效果和意义.

4.3.1 案例介绍

我们选择了一个由多个曲线拼接而成的模型, 如 fig. 4a 所示. 这个模型具有一定的复杂度, 因为它包含了多种形状和尺寸的曲线, 并且曲线之间存在一定的间隙和重叠.

我们的目标是使用拓扑优化的方法, 对这个模型进行简化和优化, 使其更加适合参数化设计和制造. 我们希望通过优化, 减少曲线的数量和节点的数量, 并消除曲线之间的缺陷和重复.

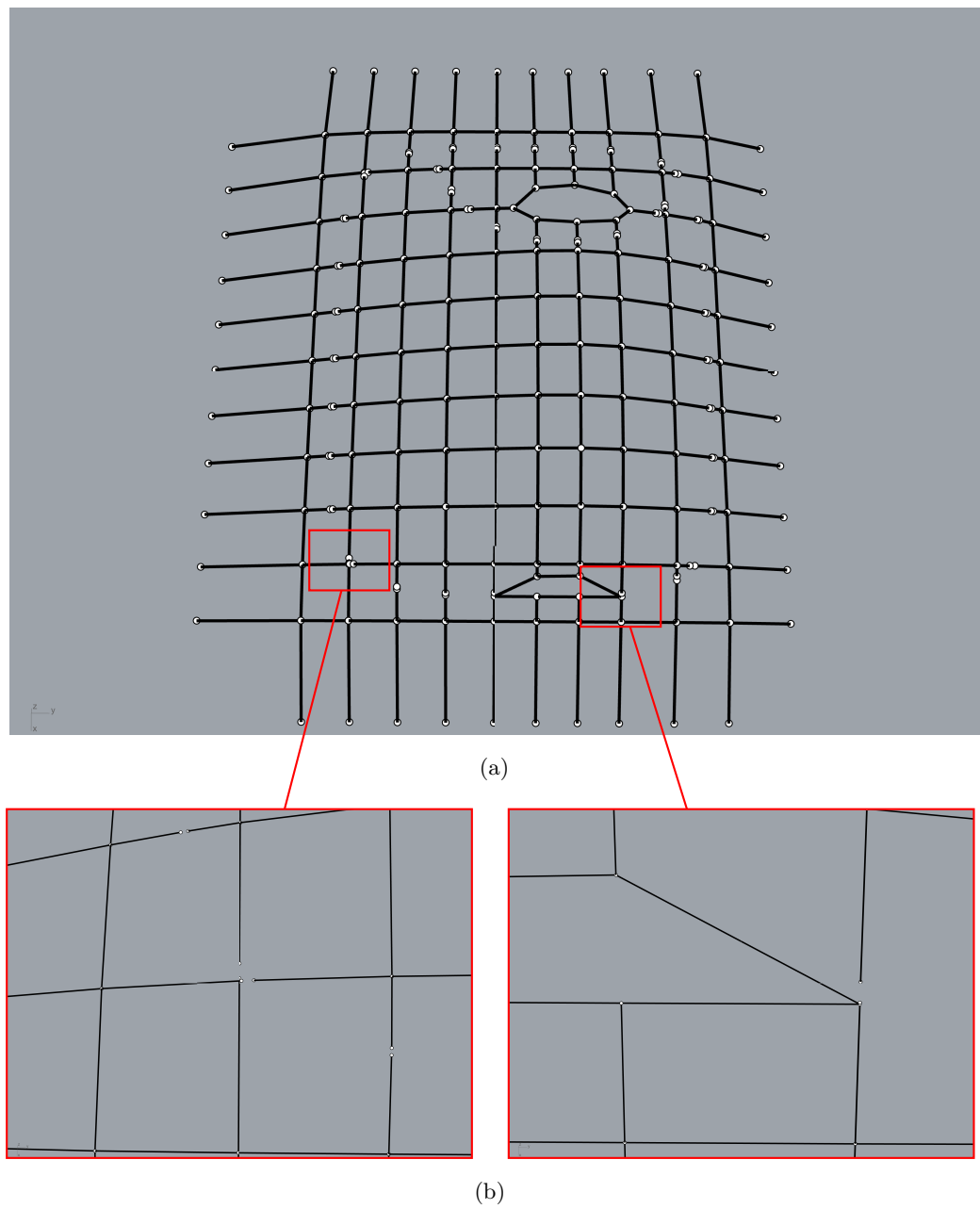


图 4: 优化前

4.3.2 优化流程

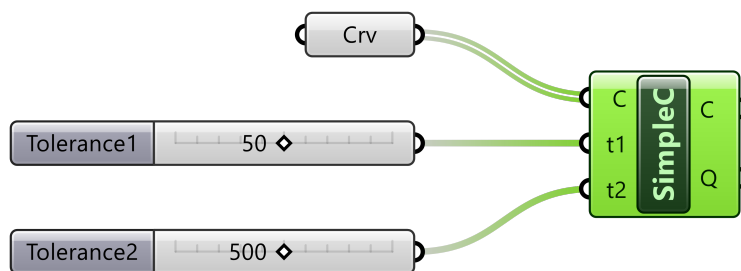


图 5: 拓扑优化流程

我们使用 Grasshopper 来实现拓扑优化的流程, 如 fig. 5 所示. 我们首先将 Rhino 中导入的模型转换为 Grasshopper 中的数据结构, 然后使用一系列的组件来对模型进行分析和处理.

4.3.3 优化结果

经过上述步骤, 我们得到了优化后的模型, 如 fig. 6a 所示. 我们可以看到, 优化后的模型具有以下特点:

- 曲线的数量从 728 个减少到 256 个, 减少了 64.8%.
- 节点的数量从 1444 个减少到 163 个, 减少了 88.7%.
- 曲线之间的缺陷和重复都被消除了, 模型更加整洁和规整.

为了更清楚地展示优化的效果, 我们选取了模型的两个局部区域, 并将其放大, 如 figs. 4b and 6b 所示. 我们可以对比优化前后的局部图, 发现优化后的曲线更加平滑和连续, 并且没有断裂和重叠的现象. 这说明我们的拓扑优化方法可以有效地修复模型中的缺陷, 并提高模型的质量.

4.3.4 优化意义

通过拓扑优化, 我们可以提高模型的质量和效率, 并为后续的参数化设计和制造提供更好的基础. 我们可以从以下几个方面来说明优化的意义:

- 简化模型可以减少计算量和存储空间, 提高运行速度和稳定性.
- 简化模型可以减少设计变量和约束条件, 提高设计灵活性和创新性.
- 简化模型可以减少制造难度和成本, 提高制造精度和质量.

综上所述, 拓扑优化是一种有效的方法, 可以帮助我们在实际案例中实现更好的参数化设计和制造. 在未来的工作中, 我们将继续探索拓扑优化的更多可能性和应用场景.

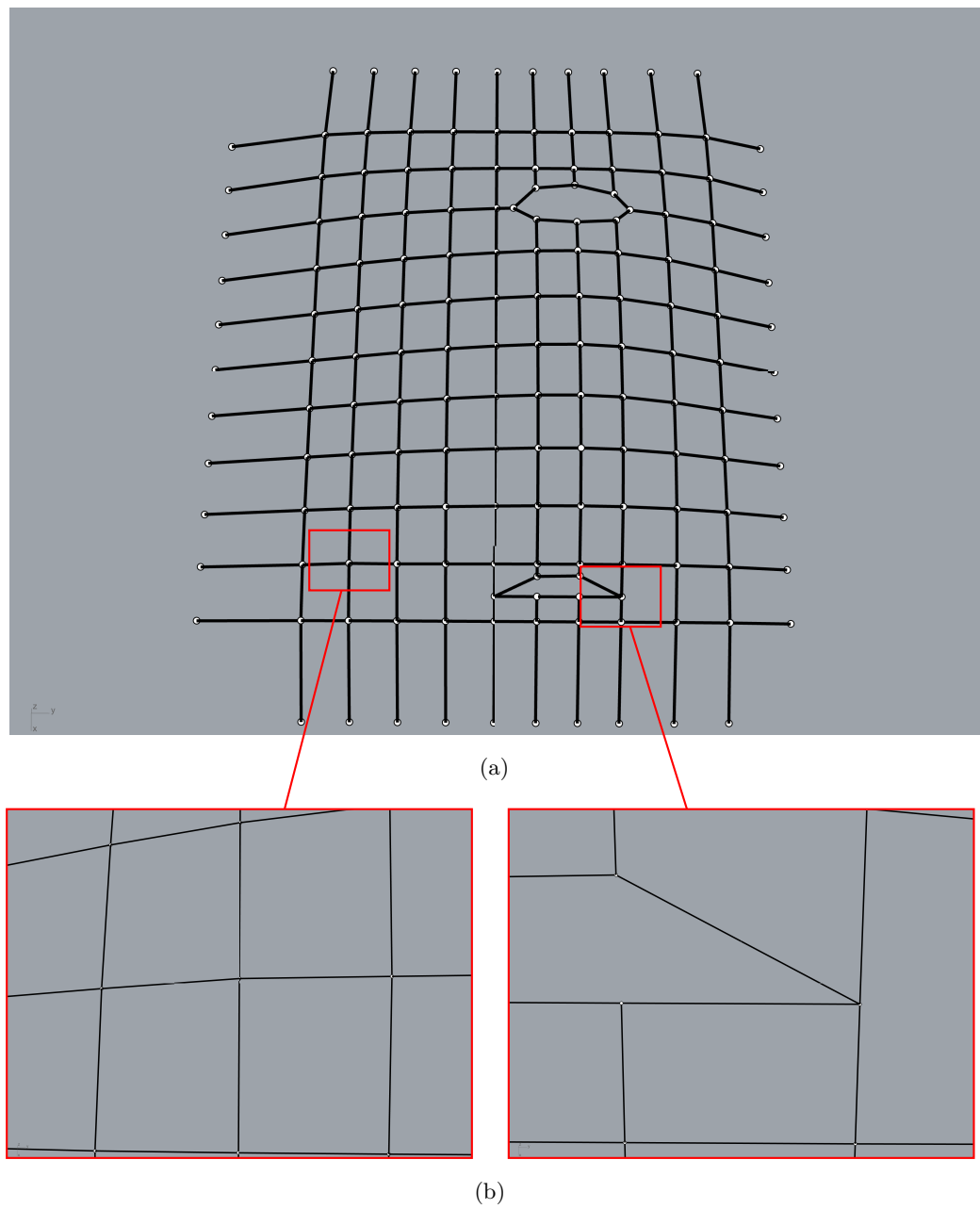


图 6: 优化后

5 结论

本文以 Rhino 和 Grasshopper 为平台, 探讨了参数化设计中的曲线打断, 容差合并, 拓扑优化等问题. 主要内容和结论如下:

- 本文提出了一种基于树形数据的曲线打断方法, 可以有效地将复杂的曲线网络分割成单独的曲线段, 并保持其拓扑关系. 该方法可以应用于参数化设计中的多种场景, 如网格生成, 曲面分割, 结构优化等.
- 本文分析了容差合并的重要性的问题, 并比较了三种容差合并算法: 朴素容差合并算法, 基于近邻搜索的容差合并算法和基于 k-d Tree 的容差合并算法. 结果表明, 基于 k-d Tree 的容差合并算法具有最高的效率, 可以在较短的时间内处理大量的点集数据. 本文还提出了一种点集的排序方法, 可以进一步提高容差合并的稳定性.
- 本文介绍了拓扑优化的重要性的方法, 并以一个实际案例为例, 展示了如何利用 Grasshopper 和 C# 插件实现拓扑优化的功能. 通过拓扑优化, 可以消除参数化设计中产生的多余或错误的元素, 提高模型的质量和可用性.

本文为参数化设计提供了一些有用的工具和技巧, 希望能够为参数化设计的发展和应用带来一些启示和帮助. 当然, 参数化设计还有许多未解决的问题和挑战, 需要进一步的研究和探索. 本文仅是一个初步的尝试.

A 附录

A.1 C# 源码

Listing 6. Graph.cs

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace HelloGrasshopper
5 {
6     internal class Edge<T> : IComparable<Edge<T>>, IEquatable<Edge<T>>
7         where T : IComparable<T>
8     {
9         public Edge(Vertex<T> from, Vertex<T> to)
10         {
11             switch (from.CompareTo(to))
12             {
13                 case -1:
14                     this.from = from;
15                     this.to = to;
16                     break;
17                 case 0:
18                     throw new Exception("Cannot create an edge from a vertex to itself.");
19                 case 1:
20                     this.from = to;
21                     this.to = from;
22                     break;
23                 default:
24                     throw new Exception("Unexpected comparison result.");
25             }
26         }
27
28         public int CompareTo(Edge<T> other)
29         {
30             switch (this.from.CompareTo(other.from))
31             {
32                 case -1:
33                     return -1;
34                 case 0:
35                     return this.to.CompareTo(other.to);
36                 case 1:
37                     return 1;
38                 default:
39                     throw new Exception("Unexpected comparison result.");
40             }
41         }
42     }
```

```

43     public bool Equals(Edge<T> other) =>
44         this.from.Equals(other.from) && this.to.Equals(other.to);
45
46     public override int GetHashCode() => this.from.GetHashCode() ^ this.to.GetHashCode();
47
48     public Vertex<T> from { get; set; }
49
50     public Vertex<T> to { get; set; }
51 }
52
53 internal class Vertex<T> : IComparable<Vertex<T>>, IEquatable<Vertex<T>>
54     where T : IComparable<T>
55 {
56     public Vertex(T data)
57     {
58         this.data = data;
59         this.adjacencies = new HashSet<Vertex<T>>();
60     }
61
62     public int CompareTo(Vertex<T> other) => this.data.CompareTo(other.data);
63
64     public bool Equals(Vertex<T> other) => this.data.Equals(other.data);
65
66     public override int GetHashCode() => this.data.GetHashCode();
67
68     public void AddAdjacency(Vertex<T> v)
69     {
70         this.adjacencies.Add(v);
71         v.adjacencies.Add(this);
72     }
73
74     public T data { get; set; }
75
76     public HashSet<Vertex<T>> adjacencies { get; set; }
77 }
78
79 internal class Graph<T>
80     where T : IComparable<T>, IEquatable<T>
81 {
82     public Graph() => this.vertices = new HashSet<Vertex<T>>();
83
84     public void AddEdge(Vertex<T> u, Vertex<T> v) => u.AddAdjacency(v);
85
86     public void AddEdge(T u, T v)
87     {
88         this.vertices.TryGetValue(new Vertex<T>(u), out Vertex<T> uu);
89         this.vertices.TryGetValue(new Vertex<T>(v), out Vertex<T> vv);
90         this.AddEdge(uu, vv);

```

```
91     }
92
93     public void AddVertex(Vertex<T> v) => this.vertices.Add(v);
94
95     public void AddVertex(T data) => this.vertices.Add(new Vertex<T>(data));
96
97     public void ContractVertex(Vertex<T> root, Vertex<T> vertex)
98     {
99         foreach (Vertex<T> u in vertex.adjacencies)
100             this.AddEdge(root, u);
101         this.RemoveVertex(vertex);
102     }
103
104     public void ContractVertices(Vertex<T> root, IEnumerable<Vertex<T>> vertices)
105     {
106         foreach (Vertex<T> u in vertices)
107             this.ContractVertex(root, u);
108     }
109
110     public void RemoveDegree2Vertex(Vertex<T> vertex)
111     {
112         if (vertex.adjacencies.Count != 2)
113             return;
114         List<Vertex<T>> adjacencies = new List<Vertex<T>>(vertex.adjacencies);
115         this.AddEdge(adjacencies[0], adjacencies[1]);
116         this.RemoveVertex(vertex);
117     }
118
119     public void RemoveDegree2Vertices()
120     {
121         List<Vertex<T>> vertices = new List<Vertex<T>>(this.vertices);
122         this.RemoveDegree2Vertices(vertices);
123     }
124
125     public void RemoveDegree2Vertices(IEnumerable<Vertex<T>> vertices)
126     {
127         foreach (Vertex<T> u in vertices)
128             this.RemoveDegree2Vertex(u);
129     }
130
131     public void RemoveIsolatedVertices()
132     {
133         List<Vertex<T>> to_remove = new List<Vertex<T>>();
134         foreach (Vertex<T> v in this.vertices)
135             if (v.adjacencies.Count == 0)
136                 to_remove.Add(v);
137         this.vertices.ExceptWith(to_remove);
138     }
```



```
139
140     public void RemoveLoops()
141     {
142         foreach (Vertex<T> v in this.vertices)
143             v.adjacencies.Remove(v);
144     }
145
146     public void RemoveVertex(Vertex<T> vertex)
147     {
148         foreach (Vertex<T> u in vertex.adjacencies)
149             u.adjacencies.Remove(vertex);
150         this.vertices.Remove(vertex);
151     }
152
153     public void RemoveVertices(IEnumerable<Vertex<T>> vertices)
154     {
155         foreach (Vertex<T> v in vertices)
156             this.RemoveVertex(v);
157     }
158
159     public HashSet<Edge<T>> edges
160     {
161         get
162         {
163             HashSet<Edge<T>> edges = new HashSet<Edge<T>>();
164             foreach (Vertex<T> u in this.vertices)
165                 foreach (Vertex<T> v in u.adjacencies)
166                     edges.Add(new Edge<T>(u, v));
167             return edges;
168         }
169     }
170
171     public HashSet<Vertex<T>> vertices { get; set; }
172 }
173 }
```

Listing 7. HelloGrasshopperInfo.cs

```
1 using System;
2
3 using Grasshopper.Kernel;
4
5 namespace HelloGrasshopper
6 {
7     public class HelloGrasshopperInfo : GH_AssemblyInfo
8     {
9         public override string Name => "HelloGrasshopper";
10     }
11 }
```

```
11     public override string Description => "Hello, Grasshopper!";
12
13     public override Guid Id => new Guid("29B84659-974B-4C9F-A97D-8973AAC2F380");
14
15     public override string AuthorName => "liblaf";
16
17     public override string AuthorContact => "rhino@liblaf.anonaddy.me";
18 }
19 }
```

Listing 8. RandomGroupCurves.cs

```
1 using System;
2 using System.Collections.Generic;
3
4 using Grasshopper.Kernel;
5 using Rhino.Geometry;
6 using Grasshopper.Kernel.Types;
7 using Grasshopper.Kernel.Data;
8
9 namespace HelloGrasshopper
10 {
11     public class RandomGroupCurves : GH_Component
12     {
13         public RandomGroupCurves()
14             : base(
15                 name: "RandomGroupCurves",
16                 nickname: "RandCurve",
17                 description: "Generate random groups of curves",
18                 category: "liblaf",
19                 subCategory: "Random"
20             ) { }
21
22         protected override void RegisterInputParams(GH_Component.GH_InputParamManager
23             ⇨ pManager)
24         {
25             pManager.AddIntegerParameter(
26                 name: "NumGroup",
27                 nickname: "N",
28                 description: "Number of groups of curves",
29                 access: GH_ParamAccess.item,
30                 @default: 2
31             );
32         }
33
34         protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager
35             ⇨ pManager)
36         {
37         }
```

```

35         pManager.AddCurveParameter(
36             name: "Curves",
37             nickname: "C",
38             description: "Random groups of curves",
39             access: GH_ParamAccess.tree
40         );
41     }
42
43     protected override void SolveInstance(IGH_DataAccess DA)
44     {
45         int num_group = 0;
46         DA.GetData(name: "NumGroup", destination: ref num_group);
47         GH_Structure<GH_Curve> curves = new GH_Structure<GH_Curve>();
48         for (int i = 0; i < num_group; ++i)
49         {
50             int num_curve = this.rand.Next(2, 4);
51             curves.AppendRange(
52                 data: this.RandomGHCurves(num_curve, max_z: 0.0),
53                 path: new GH_Path(0, i)
54             );
55         }
56         DA.SetDataTree(0, curves);
57     }
58
59     public override Guid ComponentGuid => new
60     ↪ Guid("BD94B659-1C9B-4A53-BFFB-2E9CF5BED79B");
61
62     protected double RandomDouble(double min = 0.0, double max = 1.0) =>
63     ↪ this.rand.NextDouble() * (max - min) + min;
64
65     protected Point3d RandomPoint(
66         double min_x = 0.0,
67         double min_y = 0.0,
68         double min_z = 0.0,
69         double max_x = 1.0,
70         double max_y = 1.0,
71         double max_z = 1.0
72     )
73     {
74         double x = this.RandomDouble(min_x, max_x);
75         double y = this.RandomDouble(min_y, max_y);
76         double z = this.RandomDouble(min_z, max_z);
77         return new Point3d(x, y, z);
78     }
79
80     protected Curve RandomCurve(
81         int min_num_point = 3,
82         int max_num_point = 8,

```

```
82         double min_x = 0.0,
83         double min_y = 0.0,
84         double min_z = 0.0,
85         double max_x = 1.0,
86         double max_y = 1.0,
87         double max_z = 1.0
88     )
89     {
90         List<Point3d> points = new List<Point3d>();
91         int num_points = this.rand.Next(min_num_point, max_num_point);
92         for (int i = 0; i < num_points; ++i)
93             points.Add(this.RandomPoint(min_x, min_y, min_z, max_x, max_y, max_z));
94         return Curve.CreateInterpolatedCurve(points, degree: 3);
95     }
96
97     protected GH_Curve RandomGHCurve(
98         int min_num_point = 3,
99         int max_num_point = 8,
100         double min_x = 0.0,
101         double min_y = 0.0,
102         double min_z = 0.0,
103         double max_x = 1.0,
104         double max_y = 1.0,
105         double max_z = 1.0
106     )
107     {
108         return new GH_Curve(
109             this.RandomCurve(
110                 min_num_point,
111                 max_num_point,
112                 min_x,
113                 min_y,
114                 min_z,
115                 max_x,
116                 max_y,
117                 max_z
118             )
119         );
120     }
121
122     protected List<GH_Curve> RandomGHCurves(
123         int num_curve = 2,
124         int min_num_point = 3,
125         int max_num_point = 8,
126         double min_x = 0.0,
127         double min_y = 0.0,
128         double min_z = 0.0,
129         double max_x = 1.0,
```

```
130         double max_y = 1.0,
131         double max_z = 1.0
132     )
133     {
134         List<GH_Curve> curves = new List<GH_Curve>();
135         for (int i = 0; i < num_curve; ++i)
136             curves.Add(
137                 this.RandomGHCurve(
138                     min_num_point,
139                     max_num_point,
140                     min_x,
141                     min_y,
142                     min_z,
143                     max_x,
144                     max_y,
145                     max_z
146                 )
147             );
148         return curves;
149     }
150
151     private System.Random rand = new System.Random();
152 }
153 }
```

Listing 9. RandomPoints.cs

```
1 using System;
2
3 using Grasshopper.Kernel;
4
5 using Rhino.Collections;
6 using Rhino.Geometry;
7
8 namespace HelloGrasshopper
9 {
10     public class RandomPoints : GH_Component
11     {
12         public RandomPoints()
13             : base(
14                 name: "RandomPoints",
15                 nickname: "RandPts",
16                 description: "Generate random points",
17                 category: "liblaf",
18                 subCategory: "Random"
19             ) { }
20     }
```

```
21     protected override void RegisterInputParams(GH_Component.GH_InputParamManager
    ↪ pManager)
22     {
23         pManager.AddIntegerParameter(
24             name: "NumPoints",
25             nickname: "N",
26             description: "Number of points",
27             access: GH_ParamAccess.item,
28             @default: 100
29         );
30         foreach (int i in new int[] { 0, 1 })
31         {
32             foreach (string axis in new string[] { "X", "Y", "Z" })
33             {
34                 pManager.AddAngleParameter(
35                     name: i == 0 ? $"Min{axis}" : $"Max{axis}",
36                     nickname: $"{axis}{i}",
37                     description: "",
38                     access: GH_ParamAccess.item,
39                     @default: Convert.ToDouble(i)
40                 );
41             }
42         }
43     }
44
45     protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager
    ↪ pManager)
46     {
47         pManager.AddPointParameter(
48             name: "Points",
49             nickname: "Pts",
50             description: "Random points",
51             access: GH_ParamAccess.list
52         );
53     }
54
55     protected override void SolveInstance(IGH_DataAccess DA)
56     {
57         int num_points = 0;
58         double min_x = 0.0,
59             min_y = 0.0,
60             min_z = 0.0,
61             max_x = 0.0,
62             max_y = 0.0,
63             max_z = 0.0;
64         DA.GetData("NumPoints", ref num_points);
65         DA.GetData("MinX", ref min_x);
66         DA.GetData("MinY", ref min_y);
```



```

67         DA.GetData("MinZ", ref min_z);
68         DA.GetData("MaxX", ref max_x);
69         DA.GetData("MaxY", ref max_y);
70         DA.GetData("MaxZ", ref max_z);
71         Point3dList points = new Point3dList();
72         for (int i = 0; i < num_points; ++i)
73         {
74             points.Add(this.RandomPoint(min_x, min_y, min_z, max_x, max_y, max_z));
75         }
76         DA.SetDataList("Points", points);
77     }
78
79     public override Guid ComponentGuid => new
80     ↪ Guid("166B24C0-A166-4E0D-BA85-692DD63765E5");
81
82     protected double RandomDouble(double min, double max) =>
83         this.rand.NextDouble() * (max - min) + min;
84
85     protected Point3d RandomPoint(
86         double min_x,
87         double min_y,
88         double min_z,
89         double max_x,
90         double max_y,
91         double max_z
92     )
93     {
94         double x = this.RandomDouble(min_x, max_x);
95         double y = this.RandomDouble(min_y, max_y);
96         double z = this.RandomDouble(min_z, max_z);
97         return new Point3d(x, y, z);
98     }
99
100     private System.Random rand = new System.Random();
101 }

```

Listing 10. RemoveDuplicatePoints.cs

```

1  using System;
2  using System.Collections.Generic;
3
4  using Grasshopper.Kernel;
5  using Grasshopper.Kernel.Geometry.SpatialTrees;
6
7  using Rhino.Geometry;
8
9  namespace HelloGrasshopper

```

```

10 {
11     public class RemoveDuplicatePoints : GH_Component
12     {
13         public RemoveDuplicatePoints()
14             : base(
15                 name: "RemoveDuplicatePoints",
16                 nickname: "DupPt",
17                 description: "Remove similar points from a list",
18                 category: "liblaf",
19                 subCategory: "Point"
20             ) { }
21
22         protected override void RegisterInputParams(GH_Component.GH_InputParamManager
23             ⇨ pManager)
24         {
25             pManager.AddPointParameter(
26                 name: "Points",
27                 nickname: "P",
28                 description: "list of points to clean",
29                 access: GH_ParamAccess.list
30             );
31             pManager.AddNumberParameter(
32                 name: "Tolerance",
33                 nickname: "t",
34                 description: "If any points are less than this distance, they will be
35                     ⇨ combined",
36                 access: GH_ParamAccess.item,
37                 @default: 0.01
38             );
39         }
40
41         protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager
42             ⇨ pManager)
43         {
44             pManager.AddPointParameter(
45                 name: "UniquePoints",
46                 nickname: "Q",
47                 description: "list of unique points",
48                 access: GH_ParamAccess.list
49             );
50         }
51
52         protected override void SolveInstance(IGH_DataAccess DA)
53         {
54             List<Point3d> points = new List<Point3d>();
55             double tolerance = 0.01;
56             DA.GetDataList("Points", points);
57             DA.GetData("Tolerance", ref tolerance);

```

```

55     points.Sort((Point3d a, Point3d b) => a.CompareTo(b));
56     HashSet<Point3d> point_set = new HashSet<Point3d>(points);
57     Node3d<Point3d> tree = new Node3d<Point3d>(
58         converter: (Point3d pt, out double x, out double y, out double z) =>
59         {
60             x = y = z = 0.0;
61             TreeDelegates.Point3dCoordinates(pt, ref x, ref y, ref z);
62         },
63         region: new BoundingBox(points)
64     );
65     tree.AddRange(points);
66     for (int i = 0; i < points.Count; i++)
67     {
68         if (!point_set.Contains(points[i]))
69             continue;
70         while (true)
71         {
72             List<Index3d<Point3d>> neighbors = tree.NearestItems(
73                 locus: points[i],
74                 groupSize: 2,
75                 minimumDistance: 0.0,
76                 maximumDistance: tolerance
77             );
78             if (neighbors.Count < 2)
79                 break;
80             Index3d<Point3d> neighbor = neighbors[1];
81             if (points[i].DistanceTo(neighbor.Item) > tolerance)
82                 break;
83             point_set.Remove(neighbor.Item);
84             tree.Remove(neighbor);
85         }
86     }
87     DA.SetDataList("UniquePoints", point_set);
88 }
89
90 public override Guid ComponentGuid => new
91     ↪ Guid("B0AA0E5D-0F03-4D5D-8E68-51D67791F03D");
92 }

```

Listing 11. Simplification.cs

```

1 using System;
2 using System.Collections.Generic;
3
4 using Grasshopper.Kernel;
5 using Rhino.Geometry;
6

```

```
7 namespace HelloGrasshopper
8 {
9     public class Simplification : GH_Component
10    {
11        public Simplification()
12            : base(
13                name: "Simplification",
14                nickname: "SimpleC",
15                description: "Simplify curves",
16                category: "liblaf",
17                subCategory: "Curve"
18            ) { }
19
20        protected override void RegisterInputParams(GH_Component.GH_InputParamManager
21        ↵ pManager)
22        {
23            pManager.AddCurveParameter(
24                name: "Curves",
25                nickname: "C",
26                description: "list of curves to simplify",
27                access: GH_ParamAccess.list
28            );
29            pManager.AddNumberParameter(
30                name: "Tolerance1",
31                nickname: "t1",
32                description: "If any points are less than this distance apart, they will be
33                ↵ combined",
34                access: GH_ParamAccess.item,
35                @default: 0.01
36            );
37            pManager.AddNumberParameter(
38                name: "Tolerance2",
39                nickname: "t2",
40                description: "If any points are less than this distance apart, they will be
41                ↵ combined",
42                access: GH_ParamAccess.item,
43                @default: 0.1
44            );
45        }
46
47        public override Guid ComponentGuid => new
48        ↵ Guid("E22536A3-C049-406C-A211-6BFA020EB593");
49
50        protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager
51        ↵ pManager)
52        {
53            pManager.AddCurveParameter(
54                name: "UniqueCurves",
```

```

50         nickname: "C",
51         description: "list of unique curves",
52         access: GH_ParamAccess.list
53     );
54     pManager.AddPointParameter(
55         name: "UniquePoints",
56         nickname: "Q",
57         description: "list of unique points",
58         access: GH_ParamAccess.list
59     );
60 }
61
62 protected override void SolveInstance(IGH_DataAccess DA)
63 {
64     List<Curve> curves = new List<Curve>();
65     double tolerance_1 = 0.0;
66     double tolerance_2 = 0.0;
67     DA.GetDataList(name: "Curves", list: curves);
68     DA.GetData(name: "Tolerance1", destination: ref tolerance_1);
69     DA.GetData(name: "Tolerance2", destination: ref tolerance_2);
70     Graph<Point3d> graph = new Graph<Point3d>();
71     foreach (Curve curve in curves)
72     {
73         foreach (Curve segment in curve.DuplicateSegments())
74         {
75             Point3d start = segment.PointAtStart;
76             Point3d end = segment.PointAtEnd;
77             graph.AddVertex(start);
78             graph.AddVertex(end);
79             graph.AddEdge(start, end);
80         }
81     }
82     Simplify(ref graph, tolerance_1);
83     Simplify(ref graph, tolerance_2);
84     graph.RemoveDegree2Vertices();
85     List<LineCurve> unique_curves = new List<LineCurve>();
86     foreach (Edge<Point3d> e in graph.edges)
87         unique_curves.Add(new LineCurve(from: e.from.data, to: e.to.data));
88     DA.SetDataList(paramName: "UniqueCurves", data: unique_curves);
89     List<Point3d> unique_points = new List<Point3d>();
90     foreach (Vertex<Point3d> v in graph.vertices)
91         unique_points.Add(v.data);
92     DA.SetDataList(paramName: "UniquePoints", data: unique_points);
93 }
94
95 internal void Simplify(ref Graph<Point3d> graph, double tolerance = 0.01)
96 {
97     List<Vertex<Point3d>> vertices = new List<Vertex<Point3d>>(graph.vertices);

```

```

98         vertices.Sort(
99             (Vertex<Point3d> u, Vertex<Point3d> v) =>
100             {
101                 if (u.adjacencies.Count == v.adjacencies.Count)
102                     return u.data.CompareTo(v.data);
103                 else
104                     return v.adjacencies.Count.CompareTo(u.adjacencies.Count);
105             }
106         );
107         foreach (Vertex<Point3d> u in vertices)
108         {
109             if (!graph.vertices.Contains(u))
110                 continue;
111             List<Vertex<Point3d>> to_remove = new List<Vertex<Point3d>>();
112             foreach (Vertex<Point3d> v in graph.vertices)
113             {
114                 if (u == v)
115                     continue;
116                 if (u.data.DistanceTo(v.data) < tolerance)
117                     to_remove.Add(v);
118             }
119             graph.ContractVertices(root: u, vertices: to_remove);
120         }
121         graph.RemoveLoops();
122         graph.RemoveIsolatedVertices();
123     }
124 }
125 }

```

Listing 12. SplitCurves.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  using Grasshopper.Kernel;
6  using Grasshopper.Kernel.Data;
7  using Grasshopper.Kernel.Types;
8
9  using Rhino.Geometry;
10 using Rhino.Geometry.Intersect;
11
12 namespace HelloGrasshopper
13 {
14     public class SplitCurves : GH_Component
15     {
16         public SplitCurves()
17             : base(

```

```
18         name: "SplitCurves",
19         nickname: "SplitC",
20         description: "Split curves at intersections",
21         category: "liblaf",
22         subCategory: "Curve"
23     ) { }
24
25     protected override void RegisterInputParams(GH_Component.GH_InputParamManager
26     ↪ pManager)
27     {
28         pManager.AddCurveParameter(
29             name: "Curves",
30             nickname: "C",
31             description: "Curves to split",
32             access: GH_ParamAccess.tree
33         );
34     }
35
36     protected override void RegisterOutputParams(GH_Component.GH_OutputParamManager
37     ↪ pManager)
38     {
39         pManager.AddCurveParameter(
40             name: "Curves",
41             nickname: "C",
42             description: "Split curves",
43             access: GH_ParamAccess.tree
44         );
45     }
46
47     protected override void SolveInstance(IGH_DataAccess DA)
48     {
49         DA.GetDataTree(name: "Curves", tree: out GH_Structure<GH_Curve> curves);
50         GH_Structure<GH_Curve> result = new GH_Structure<GH_Curve>();
51         foreach (GH_Path path in curves.Paths)
52         {
53             for (int i = 0; i < curves[path].Count; ++i)
54             {
55                 Curve curve = curves[path][i].Value;
56                 List<double> parameters = new List<double>();
57                 foreach (GH_Curve gh_cutter in curves.AllData(skipNulls: true))
58                 {
59                     Curve cutter = gh_cutter.Value;
60                     if (cutter == curve)
61                         continue;
62                     CurveIntersections intersections = Intersection.CurveCurve(
63                         curve,
64                         cutter,
65                         tolerance: 0.0,
```

```

64         overlapTolerance: 0.0
65     );
66     parameters.AddRange(
67         intersections.Select(intersection => intersection.ParameterA)
68     );
69 }
70 parameters.Sort();
71 parameters = parameters
72     .Prepend(curve.Domain.Min)
73     .Append(curve.Domain.Max)
74     .ToList();
75 List<GH_Curve> segments = new List<GH_Curve>();
76 for (int j = 0; j + 1 < parameters.Count; ++j)
77 {
78     double t0 = parameters[j];
79     double t1 = parameters[j + 1];
80     segments.Add(new GH_Curve(curve.Trim(t0, t1)));
81 }
82 result.AppendRange(data: segments, path: path.AppendElement(i));
83 }
84 }
85 DA.SetDataTree(paramIndex: 0, tree: result);
86 }
87
88 public override Guid ComponentGuid => new
89     ↪ Guid("42F95841-B049-4F0E-B9C6-3B61EADF2DBF");
90 }

```

❖ A.2 Python 源码

Listing 13. remove_duplicate_points_naive.py

```

1  """
2  remove_duplicate_points
3
4  Removes similar points from a list
5
6  Inputs:
7      P (Point) : list of points to clean
8      t (Number) : points closer than this distance will be combined
9
10 Output:
11     Q (Point) : list of unique points
12 """
13
14 import itertools
15

```



```

16 import rhinoscriptsyntax as rs
17
18
19 def point_less(u, v):
20     u_coordinates = rs.PointCoordinates(u)
21     v_coordinates = rs.PointCoordinates(v)
22     return u_coordinates < v_coordinates
23
24
25 removed = set()
26 for u, v in itertools.combinations(P, 2):
27     if u in removed or v in removed:
28         continue
29     if not rs.PointCompare(u, v, tolerance=t):
30         continue
31     removed.add(v if point_less(u, v) else u)
32 Q = [u for u in P if u not in removed]

```

Listing 14. remove_duplicate_points.py

```

1  """
2  remove_duplicate_points
3
4  Removes similar points from a list
5
6  Inputs:
7      P (Point) : list of points to clean
8      t (Number) : points closer than this distance will be combined
9
10 Output:
11     Q (Point) : list of unique points
12 """
13
14 import itertools
15 import subprocess
16 from array import array
17
18 import rhinoscriptsyntax as rs
19
20 SRC = """
21 import sys
22 from array import array
23
24 import numpy as np
25 from scipy.spatial import KDTree
26
27 raw: array = array("f")
28 raw.frombytes(sys.stdin.buffer.read())

```

```
29 points: np.ndarray = np.array(raw[:-1]).reshape(-1, 3)
30 tolerance: float = raw[-1]
31 kdtree: KDTree = KDTree(data=points)
32 results: set[int] = set(range(points.shape[0]))
33 for i in np.lexsort(keys=points[:, :-1].transpose()):
34     if i not in results:
35         continue
36     duplicates: set[int] = (
37         set(kdtree.query_ball_point(x=points[i], r=tolerance)) & results
38     )
39     min_id: int = min(duplicates, key=lambda x: tuple(points[x]))
40     results -= set(duplicates)
41     results.add(min_id)
42 print(results)
43 """
44
45 arr = array("f")
46 arr.fromlist(itertools.chain(*map(rs.PointCoordinates, P)))
47 arr.append(t)
48 startupinfo = subprocess.STARTUPINFO()
49 startupinfo.dwFlags = subprocess.STARTF_USESTDHANDLES | subprocess.STARTF_USESHOWWINDOW
50 process = subprocess.Popen(
51     args=["python", "-c", SRC],
52     stdin=subprocess.PIPE,
53     stdout=subprocess.PIPE,
54     stderr=subprocess.PIPE,
55     startupinfo=startupinfo,
56 )
57 stdout, stderr = process.communicate(arr.tostring())
58 output = eval(stdout)
59 Q = [P[i] for i in output]
```