|  |  |
| --- | --- |
| 论文题目： | 基于WebGL的三维地质建模及可视化方法研究 |

**目录**

[**1 绪论** 1](#_Toc152253748)

[1.1 研究背景及意义 2](#_Toc152253750)

[1.2 国内外研究现状 3](#_Toc152253751)

1.2.1 三维地质建模研究现状 3

1.2.2 三维地质可视化研究现状 3

[1.3 研究内容与技术路线 3](#_Toc152253751)

1.3.1 研究内容 3

1.3.2 技术路线 3

[**2 三维地质模型构建** 2](#_Toc152253749)

[2.1 Delaunay 三角剖分方法 2](#_Toc152253750)

2.1.1 遵循Delaunay准则的约束三角剖分 3

2.1.2 三维地质模型数据结构 3

[2.2 钻孔模型构建 3](#_Toc152253751)

[2.3 断层模型构建 3](#_Toc152253751)

2.3.1 相交断层模型的构建方法 3

2.3.2 断层模型网格化 3

[2.4 地层模型构建 3](#_Toc152253751)

2.4.1 CDT剖分数据源提取 3

2.4.2 地层数据网格化 3

2.5 本章小结

[**3 基于WebGL可视化方法** 5](#_Toc152253752)

[3.1 WebGL可视化技术 5](#_Toc152253753)

[3.2 三维场景构建 5](#_Toc152253754)

[3.3 模型渲染方法 5](#_Toc152253754)

3.3.1 **模型纹理uv计算方法** 3

3.3.2 模型可视化 3

[3.4 交互功能 5](#_Toc152253754)

3.4.1 射线追踪 3

3.4.2 模型控制 3

3.4.3 巷道漫游 3

3.4.4 点选功能 3

[**4 系统设计与实现** 6](#_Toc152253756)

[4.1 系统架构设计 6](#_Toc152253757)

4.1.1 体系结构 3

4.1.2 技术流程 3

[4.2 系统功能实现 7](#_Toc152253758)（待定）

4.2.1 地质模型可视化 3

4.2.2 地层层间距计算 3

4.2.3注浆钻孔轨迹检测 3

[**5 应用实例** 8](#_Toc152253759)

[5.1 研究区概况 2](#_Toc152253750)

[5.2 三维地质建模 3](#_Toc152253751)

[5.3 可视化分析 2](#_Toc152253750)

[5.4 (本章小结) 3](#_Toc152253751)

[**6 结论与展望** 8](#_Toc152253760)

[6.1 结论 2](#_Toc152253750)

[6.2 展望 3](#_Toc152253751)

[**参考文献** 9](#_Toc152253762)

1. 绪论

1.1研究背景及意义

随着科学技术的不断发展，三维地质建模在地质勘探、工程地质和地下资源开发等领域得到了广泛应用。传统的二维地质图已难以满足复杂地质体的精确表达和分析需求，三维地质建模不仅能够更加直观、准确地呈现地质体的空间形态，还可以提供更丰富的地质信息，这对于提升地质工程设计的精度和支持科学决策具有重要作用。

基于 WebGL 的三维地质建模与可视化方法代表了当前技术发展的一个重要方向。WebGL 作为一种基于浏览器的开放式图形标准，通过集成 GPU 加速的三维渲染技术，使得用户无需安装插件即可在浏览器中实时渲染复杂的三维图像。与传统的桌面应用程序相比，这种方法具有跨平台兼容性强、部署维护简便、实时交互性高等优势。通过 WebGL 技术，用户可以在任何具备浏览器的设备上运行地质建模与可视化平台，实现跨操作系统和设备的无缝访问和使用，大大提高了应用的灵活性和可扩展性。

相比传统的 C/S（客户端-服务器）架构，B/S（浏览器-服务器）架构在地质建模系统中的应用具有明显的优势。首先，B/S 架构基于浏览器操作，能够适配多种操作系统，避免了客户端安装与维护的复杂性，极大地降低了软件的部署和维护成本。其次，B/S 架构通过将核心计算任务转移到服务器端，使得用户即便是在较为轻量的客户端设备上，也可以顺畅地加载和操作大规模的三维地质模型。此外，B/S 架构还支持多用户远程协作，通过互联网实现地质信息的即时共享与交流，这在大型地质项目的协作中尤为重要。

本研究的意义不仅体现在技术层面上，还具备广泛的学术与应用价值。从学术角度来看，本文探索了基于 WebGL 和 B/S 架构的三维地质建模与可视化方法，为三维地质建模提供了新的技术路径。它为相关领域的研究人员开辟了一种新的数据处理和展示方式，极大地推动了该领域信息化、智能化的进程。从实际应用角度来看，本研究能够有效应用于地质勘探、地下工程、矿山开发等领域，突破了传统软件在安装、升级以及跨平台使用中的局限性，提升了系统的应用效率与用户体验。

未来，随着 Web 技术的进一步发展，基于 WebGL 的三维地质建模平台有望进一步扩展其功能和应用场景。例如，可以将虚拟现实（VR）与增强现实（AR）技术结合，提供更加沉浸式的地质模型展示与交互功能。这将为地质工程、城市规划、建筑设计等多个领域提供更丰富的应用前景，并推动三维地质建模在行业中的深度应用与广泛传播。

1.2国内外研究现状

随着信息技术的快速发展，三维地质建模已成为地质领域的重要研究方向，并在全球范围内得到了广泛应用。该技术在地质勘探、地下资源开发、工程建设等领域发挥着关键作用。然而，由于技术水平、研究重点和应用需求的不同，各国和地区在三维地质建模及可视化技术的研究与实践上存在显著差异。以下将分别对三维地质建模技术和三维地质可视化技术的国内外研究现状进行综述。

1.2.1 **三维地质建模技术的国内外研究现状**

三维地质建模技术是地质信息化的重要组成部分，经过多年的发展，国内外学者围绕建模方法、数据处理和系统实现等方面进行了广泛研究。在国内，三维地质建模技术主要聚焦于结合钻孔、地质剖面和地震反射等数据的建模方法。基于三角网（TIN）和规则网格的建模方法被广泛应用，具有较好的建模效率和可操作性。然而，面对复杂地质结构，模型的精度和动态调整能力仍存在较大挑战。近年来，国内开发了一些具有自主知识产权的地质建模软件，例如三维地质体建模系统（3DGIS），已在矿产资源评估和工程地质等领域中取得了一定的应用成果。然而，与国际先进技术相比，国内建模工具在大规模数据处理、用户交互设计和智能化程度方面仍显不足。

相比之下，国外在三维地质建模领域起步较早，研究更加系统且深入。欧美发达国家开发了诸如GOCAD、Petrel和Leapfrog等成熟的地质建模软件，这些工具集成了多种先进算法，如隐函数法、基于多分辨率的建模技术和智能化的模型优化方法，在处理复杂地质构造和大数据方面表现突出。此外，国外学者注重结合云计算和大数据分析技术，以提高建模效率并增强对实时数据的适应性。这种技术趋势不仅显著提高了建模的精度和效率，也为解决动态数据更新和模型演化问题提供了新思路。

1.2.2. 三维地质可视化技术的国内外研究现状

三维地质可视化技术作为地质信息表达的核心手段，为地质数据的展示与交互提供了重要支持。在国内，三维地质可视化技术的研究起步相对较晚，但近年来随着WebGL等新技术的引入，相关研究取得了显著进展。基于WebGL的三维地质信息系统已逐步应用于在线地质展示与分析，通过实时渲染和多源数据融合，实现了地质模型的动态交互与可视化。与此同时，国内学者在提升可视化效率和图像质量方面也做了大量工作，例如采用基于体绘制和光线追踪的优化算法，显著改善了模型渲染的清晰度和效率。然而，国内在高性能可视化渲染、复杂场景处理和用户体验优化方面仍存在不足，尤其是Web端的实时交互性能还有较大提升空间。

国外在三维地质可视化领域具有领先优势，尤其是在实时渲染、虚拟现实（VR）和增强现实（AR）技术的融合应用方面。近年来，国外利用WebGL开发了诸如Cesium和Three.js等可视化框架，这些框架为地质数据的三维展示提供了强大的技术支持，能够在低硬件资源环境下实现高质量的渲染和交互。此外，VR和AR技术的引入极大提升了可视化的沉浸感和场景还原能力。例如，通过VR设备，用户可以进入虚拟地质场景进行交互式观测，而AR技术则使得三维地质模型可以叠加到真实场景中，增强了可视化的应用价值。当前，国外研究还在探索如何结合人工智能技术，通过自动化的模型渲染和细节优化进一步提升可视化效率，展现出较为明显的技术前沿优势。

1.3研究内容与技术路线

1.3.1 研究内容

本研究旨在基于 WebGL 技术开发一个三维地质建模与可视化平台，利用现代 Web 技术实现复杂地质体的建模与可视化表达，并提供丰富的交互功能和轻量化、跨平台的解决方案。研究内容涵盖地质数据的处理与建模、三维可视化技术的实现、交互功能的开发以及平台性能优化。

在建模方面，本研究以钻孔数据为核心数据源，聚焦于地层、注浆钻孔、地质剖面的三维建模。通过对钻孔数据的分析与整合，采用动态数据融合和多分辨率建模技术，逐层重建复杂地质结构。同时，本研究还将实现巷道、勘探线等辅助工程结构的精细化建模，为资源勘探、地下工程设计等领域提供数据支持和技术保障。建模过程中，将注重算法的高效性和准确性，确保生成模型能真实反映地质条件和工程特性。

在可视化实现中，本研究基于 WebGL 技术，采用 Three.js 库作为开发工具。Three.js 作为 WebGL 的高层封装，能够显著降低开发复杂度，同时支持高性能三维渲染。平台将通过 GPU 加速实现地质模型的实时渲染，并优化复杂场景的渲染性能，确保在多层次地质数据叠加的情况下仍具备流畅的可视化效果。研究将设计巷道、地层结构和钻孔等数据的分层渲染功能，用户可通过显隐控制动态管理各类图层，快速切换视角，满足不同应用场景的需求。

交互功能是本研究的重点之一。平台将实现三维模型的旋转、缩放、剖切操作，以及对关键地质信息的查询功能。用户可以通过交互界面对地质模型进行直观的分析，结合多角度视图和属性数据展示，深入理解地质结构与工程布局。此外，系统将支持图层显隐控制、模型分块加载和分级渲染，确保用户能够高效操作并聚焦于目标区域。

本研究还将充分发挥 Web 技术的优势，构建一个轻量化、跨平台的三维地质可视化解决方案。通过 Web 端访问方式，用户无需安装额外的软件，只需通过主流浏览器即可快速加载并使用系统。Web 端可视化具有较低的硬件依赖性，同时能够实现高效的数据共享与协作，大幅降低系统的使用门槛。此外，平台支持多设备兼容，可在桌面端、移动端等多种设备上运行，进一步拓展了地质数据应用的场景。

综上所述，本研究通过构建基于 WebGL 的三维地质建模与可视化平台，为地层建模、钻孔分析、剖面研究、巷道设计以及勘探线的可视化提供了技术支持。平台的开发不仅能够满足地质信息的表达与交互需求，还能够为资源开发、工程设计和科学研究提供直观高效的技术工具，推动地质信息化向更高效、更便捷的方向发展。

1.3.2 技术路线

本研究的技术路线结合了三维地质建模与现代 Web 前端技术的特点，经历了从原生 HTML 开发到基于 Vue3 框架的系统升级过程。整体技术路线分为两大阶段，分别为系统的初始开发阶段和后续的系统升级与优化阶段。

1. 初始开发阶段

在系统的初始开发阶段，主要采用原生 HTML 和 JavaScript 技术，基于 WebGL 的 Three.js 库进行三维地质建模与可视化实现。Three.js 是 WebGL 的高层封装库，提供了丰富的 3D 图形绘制功能，简化了三维图形的开发复杂度。在这一阶段，系统完成了以下功能的实现：

三维地质模型的构建与渲染：基于钻孔数据实现地层、注浆钻孔、剖面的建模；通过 Three.js 提供的几何体与材质接口，将地质结构以图形化形式展示。

巷道与勘探线可视化：通过辅助工程数据构建巷道和勘探线模型，并在三维场景中直观呈现。

交互功能的初步实现：利用 Three.js 的交互功能实现了三维模型的旋转、缩放和剖切操作，支持用户进行基本的地质数据分析。

图层显隐控制：在模型渲染基础上实现了对不同地质图层的显隐切换，用户可根据需求动态调整可视化内容。

虽然原生 HTML 和 JavaScript 的开发方式在初期能够快速实现功能，但随着系统复杂度的提升，代码的可维护性、扩展性和开发效率逐渐受到限制。因此，系统需要进行技术升级。

2. 系统升级阶段

为应对系统复杂度增加以及用户体验优化的需求，本研究在后续开发中引入了 Vue3 框架，并结合 Pinia 状态管理和 Element-UI 组件库，对系统进行了全面的升级优化。

基于 Vue3 的组件化开发：通过 Vue3 实现了系统的模块化重构，将模型展示、数据管理和交互功能分离成独立的组件，大幅提升了代码的可维护性和复用性。同时，Vue3 的响应式数据绑定机制显著提高了界面的动态更新效率。

使用 Pinia 实现状态管理：系统的状态管理采用了 Pinia 替代传统的手动数据传递方式，通过集中管理地质数据、图层状态和用户操作，简化了复杂场景下的数据处理逻辑。Pinia 的轻量化设计和 Vue3 的深度集成，为系统开发提供了更高的灵活性和性能。

Element-UI 提升用户界面体验：在前端界面设计中，引入了 Element-UI 组件库，提供了美观统一的交互组件，包括表单、滑块、对话框等，大幅优化了系统的用户体验。

优化 Three.js 渲染性能：在模型渲染过程中，进一步优化了 Three.js 的性能，例如通过分块加载和视距裁剪技术减少渲染负载，确保系统在处理复杂地质场景时仍能保持流畅的运行体验。

3.技术路线总结

通过技术路线的两阶段演进，系统逐步实现了从功能性实现到高性能、易扩展平台的转变。初期利用原生 HTML 和 Three.js 实现了基础功能，而后通过引入 Vue3、Pinia 和 Element-UI，对系统进行了模块化和性能优化，形成了一个具备高效开发、易于维护和良好用户体验的三维地质建模与可视化平台。这种技术路线不仅满足了当前的研究需求，也为系统的进一步扩展与优化奠定了坚实的基础。

1. 三维地质模型构建

## 2.1Delaunay三角剖分方法

### 2.1.1 遵循Delaunay准则的约束三角剖分（公式）

1. 离散点的普通德劳内三角剖分

德劳内三角剖分通过 增量插入 算法实现。主要步骤包括：

• 初始化：构建一个初始三角形，它包含所有要处理的点。这个初始三角形通常由三个虚拟点组成，包围所有的离散数据点。

• 点的插入：通过增量方式逐个插入离散的点。在每次插入时，确定该点所在的三角形，然后调整邻近的三角形，确保所有三角形满足德劳内准则。Tinfour 使用 Bowyer-Watson 算法 来插入新点。

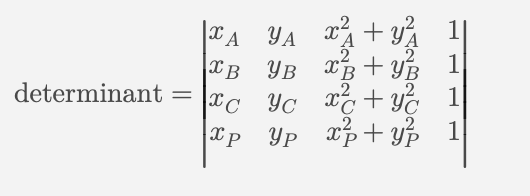
Bowyer-Watson 算法的核心逻辑：

1. 找到包含新点的三角形。

2. 将该三角形移除，并生成一个新三角形组，这些三角形与新点相连。

3. 检查新生成的三角形是否满足德劳内外接圆准则，如果不满足，则通过边交换来调整。

外接圆准则的数学公式：



当行列式的值大于 0 时，点 P 位于三角形 triangle ABC 的外接圆内，不满足德劳内条件，需要进行边交换。

**2. 添加约束边的处理**

Tinfour 中的 addConstraints() 方法允许用户将约束边（如断层、道路等地形边界）插入到现有的三角网中。处理约束边时的步骤如下：

1. 检测相交三角形：首先找到与约束边相交的所有三角形。系统会识别所有穿过或相交于约束边的三角形。

2. 分割相交三角形：在不插入 Steiner 点的情况下，约束边直接分割相交的三角形。通过边交换，保证新的三角形仍然满足约束边。

在不插入 Steiner 点的前提下，约束边的处理依赖于局部优化算法，确保插入约束边后不会破坏原始数据的几何结构。

3. 约束边与三角形边相交的处理

在 Tinfour 中，约束边与现有的三角形边相交时，可能会破坏原有的德劳内结构。在这种情况下，Tinfour 提供了以下解决方案：

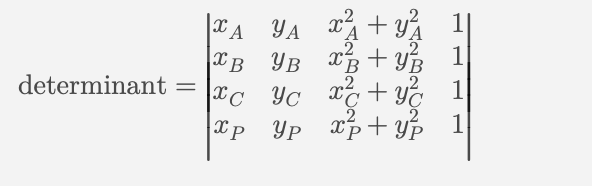
1. 边交换：当约束边与三角形边相交时，系统会通过边交换操作，调整三角形的连接方式，确保约束边不被破坏。

2. 中点插入（可选）：如果允许插入 Steiner 点，系统会在相交处插入中点，将约束边进一步细化。然而，在不允许插入 Steiner 点的情况下，系统只会通过调整已有的边来处理相交问题。

4. 数学公式

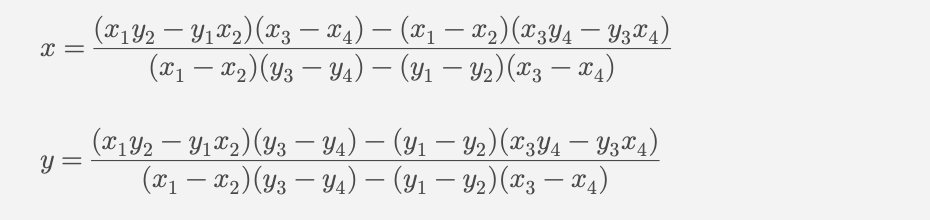
（1）外接圆条件公式

外接圆准则用于判断新点是否破坏了三角网的德劳内性。公式如下：



若行列式值大于 0，则点 $P$ 位于三角形 $\triangle ABC$ 的外接圆内，此时需要通过边交换来调整三角形。

（2）约束边与三角形边相交检测公式

约束边与三角形边相交时，需要进行几何计算来确定相交点。假设三角形边的端点为 A(x\_1, y\_1) 和 B(x\_2, y\_2)，约束边的端点为 C(x\_3, y\_3) 和 D(x\_4, y\_4)，则相交点 (x, y) 的计算公式为：

该公式用于计算约束边与三角形边的交点，确保约束边能够正确插入到三角网中。

### 2.1.2 三维地质模型数据结构（包括地层、断层、钻孔模型等等）

## 2.2断层模型构建（公式）

### 2.2.1 相交断层模型的构建方法（公式）创新点

**1. 两点之间的距离公式**

用于计算两点之间的直线距离。

**例如 comditance 函数：**

double distance = Math.sqrt((X-X1)\*(X-X1) + (Y-Y1)\*(Y-Y1));

**两点 (X, Y) 和 (X1, Y1) 之间的距离公式：**

{"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>d</mi><mo>=</mo><msqrt><msup><mfenced><mrow><mi>X</mi><mo>-</mo><mi>X</mi><mn>1</mn></mrow></mfenced><mn>2</mn></msup><mo>+</mo><msup><mfenced><mrow><mi>Y</mi><mo>-</mo><mi>Y</mi><mn>1</mn></mrow></mfenced><mn>2</mn></msup></msqrt></mstyle></math>","origin":"MathType for Microsoft Add-in"}

**2.点到直线的距离公式**

用于计算某个点到一条直线的距离。公式如下：

double A = Y2 - Y1;

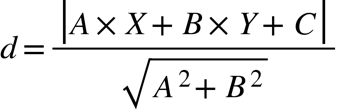
double B = X1 - X2;

double C = X2 \* Y1 - X1 \* Y2;

distance = Math.abs(A \* X + B \* Y + C) / Math.sqrt(A \* A + B \* B);

```

解释：给定两点 (X1, Y1) 和 (X2, Y2)，表示一条直线，和一个点 (X, Y)，该点到这条直线的距离公式为：



其中：

- A = Y2 - Y1

- B = X1 - X2

- C = X2 \* Y1 - X1 \* Y2

**3. 两条直线的交点公式**

用于计算两条直线的交点。该公式在intersection函数中被实现：

double x = (line2.b - line1.b) / (line1.k - line2.k)

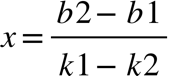
double y = x \* line1.k + line1.b

-两条直线的方程分别为：

- y = k1 \* x + b1

- y = k2 \* x + b2

它们的交点坐标 (x, y) 的计算公式为：

，{"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>y</mi><mo>=</mo><mi>k</mi><mn>1</mn><mo>&#xD7;</mo><mi>x</mi><mo>+</mo><mi>b</mi><mn>1</mn></mstyle></math>","origin":"MathType for Microsoft Add-in"}

**4. 判定两条线段是否相交**

intersection 函数的主要功能是计算两条线段的交点。

通过判断两条线段是否相交，以及如果相交，求出它们的交点坐标。

1. 输入和转换

Point[] points = checkAndConvertIntoPoint(start1\_, end1\_, start2\_, end2\_);

Point start1 = points[0];

Point end1 = points[1];

Point start2 = points[2];

Point end2 = points[3];

- 输入的 `start1\_`, `end1\_`, `start2\_`, `end2\_` 是两个线段的起点和终点，以数组形式表示 `[x, y]`。

函数首先将这些数组转换成 `Point` 对象（假设 `Point` 是表示二维点的类），然后将两个线段的起点和终点分别赋值给 `start1`, `end1`, `start2`, `end2`。

2. 封装成直线对象

Line line1 = new Line(start1, end1);

Line line2 = new Line(start2, end2);

- 将两个线段封装成 `Line` 对象，便于后续计算直线的\*\*斜率\*\*（`k`）和\*\*截距\*\*（`b`）。可以假设 `Line` 类根据给定的起点和终点自动计算出斜率和截距。

3. 处理特殊情况 1：斜率为无穷大的情况（垂直线）：

if (line1.k == Integer.MAX\_VALUE || line2.k == Integer.MAX\_VALUE) {

// 斜率都不存在时，说明两条直线都是垂直线，需要判断是否重合

}

如果其中一条线的斜率是无穷大（即垂直线），函数会首先处理这种特殊情况：

- 如果两条线都垂直，它们可能是\*\*同一条线\*\*（共线情况）。通过判断两条垂直线的 `b` 值是否接近来判断它们是否重合（`b` 表示垂直线的 x 值）。

- 如果重合，且起点、终点有交集（即 `isBetween` 函数判断的交点在两线段的范围内），则返回交点的坐标。

如果只有一条直线是垂直线，交点会是垂直线的 x 坐标与另一条直线的 y 坐标。

4. 处理特殊情况 2：两条线平行：

} else if (Math.abs(line1.k - line2.k) <= epslion) {

// 斜率相等，判断是否重合

}

- 如果两条直线的斜率相等（即平行），函数会检查它们的截距 `b` 是否也相等。如果截距相等，且线段的端点在另一条线段的范围内，则返回交点。

- 如果斜率相同，但截距不同，说明两条线段是平行且不相交的，返回空数组。

5. 处理一般情况：求交点

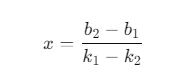
double x = (line2.b - line1.b) / (line1.k - line2.k);

double y = x \* line1.k + line1.b;

intersection = new Point(x, y);

- 如果两条直线的斜率不相等，使用交点公式求出它们的交点：

- 交点 x 坐标为：



- 交点 y 坐标为：



- 将计算出来的交点封装成 `Point` 对象。

6. \*\*判断交点是否在线段上\*\*：

if (isBetween(start1, intersection, end1) && isBetween(start2, intersection, end2)) {

return new double[]{intersection.x, intersection.y};

}

- 即使找到了交点，还需要判断该交点是否位于两条线段的范围内（线段的起点和终点之间）。这是通过 `isBetween` 函数完成的：

- `isBetween` 函数检查交点的 x 和 y 坐标是否在每条线段的起点和终点之间。

- 如果交点在两条线段的范围内，返回交点的坐标，否则返回空数组，表示两条线段不相交。

`intersection` 函数考虑了以下三种情况：

1. 特殊情况：两条线段中有一条是垂直线。

2. 特殊情况：两条线段平行（斜率相等）。

3. 一般情况：两条线段的斜率不相等，通过交点公式计算交点。

在所有情况下，函数都进一步判断交点是否位于两条线段的范围内，最终返回交点的坐标或空数组表示不相交。

### 2.2.2 断层模型网格化

## 2.3断层模型构建

### 2.3.1 相交断层模型的构建方法（\*\*具体计算的内容、方法名）（公式）

**上下盘**

主要集中在计算平面方程和计算点的坐标位置上。

**1. 平面方程的计算：**

在 get\_panel 方法中，计算了平面方程的系数 , ，这是计算一个通过三点的平面方程：

• **计算平面方程系数**：

• {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>a</mi><mo>=</mo><mfenced><mrow><mfenced><mrow><mi>p</mi><mn>2</mn><mo>.</mo><mi>y</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>y</mi></mrow></mfenced><mo>&#xD7;</mo><mfenced><mrow><mi>p</mi><mn>3</mn><mo>.</mo><mi>z</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>z</mi></mrow></mfenced><mo>-</mo><mfenced><mrow><mi>p</mi><mn>2</mn><mo>.</mo><mi>z</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>z</mi></mrow></mfenced><mo>&#xD7;</mo><mfenced><mrow><mi>p</mi><mn>3</mn><mo>.</mo><mi>y</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>y</mi></mrow></mfenced></mrow></mfenced></mstyle></math>","origin":"MathType for Microsoft Add-in"}

• {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>b</mi><mo>=</mo><mfenced><mrow><mfenced><mrow><mi>p</mi><mn>2</mn><mo>.</mo><mi>z</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>z</mi></mrow></mfenced><mo>&#xD7;</mo><mfenced><mrow><mi>p</mi><mn>3</mn><mo>.</mo><mi>x</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>x</mi></mrow></mfenced><mo>-</mo><mfenced><mrow><mi>p</mi><mn>2</mn><mo>.</mo><mi>x</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>x</mi></mrow></mfenced><mo>&#xD7;</mo><mfenced><mrow><mi>p</mi><mn>3</mn><mo>.</mo><mi>z</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>z</mi></mrow></mfenced></mrow></mfenced></mstyle></math>","origin":"MathType for Microsoft Add-in"}

• {"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>a</mi><mo>=</mo><mfenced><mrow><mfenced><mrow><mi>p</mi><mn>2</mn><mo>.</mo><mi>x</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>x</mi></mrow></mfenced><mo>&#xD7;</mo><mfenced><mrow><mi>p</mi><mn>3</mn><mo>.</mo><mi>y</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>y</mi></mrow></mfenced><mo>-</mo><mfenced><mrow><mi>p</mi><mn>2</mn><mo>.</mo><mi>y</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>y</mi></mrow></mfenced><mo>&#xD7;</mo><mfenced><mrow><mi>p</mi><mn>3</mn><mo>.</mo><mi>x</mi><mo>-</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>x</mi></mrow></mfenced></mrow></mfenced></mstyle></math>","origin":"MathType for Microsoft Add-in"}

• **平面方程的常数项**：

{"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>d</mi><mo>=</mo><mo>-</mo><mfenced><mrow><mi>a</mi><mo>&#xD7;</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>x</mi><mo>+</mo><mi>b</mi><mo>&#xD7;</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>y</mi><mo>+</mo><mi>c</mi><mo>&#xD7;</mo><mi>p</mi><mn>1</mn><mo>.</mo><mi>z</mi></mrow></mfenced></mstyle></math>","origin":"MathType for Microsoft Add-in"}

以上计算结果形成一个平面方程：

{"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>a</mi><mi>x</mi><mo>+</mo><mi>b</mi><mi>y</mi><mo>+</mo><mi>c</mi><mi>z</mi><mo>+</mo><mi>d</mi><mo>=</mo><mn>0</mn></mstyle></math>","origin":"MathType for Microsoft Add-in"}

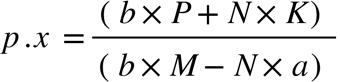
**2. Z 坐标的计算：**

在代码中直接更新了p.z的值：

p.z = p3.z + Q

**3. X 坐标的计算：**

在 get\_panel 方法中，根据其他参数计算x坐标：



其中：

• M = p3.x - p1.x

• N = p3.y - p1.y

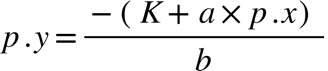
• K = c \* p.z + d

• G = (p3.z - p1.z) \* (p3.z - p.z)

• P = G + M\* p3.x + N\*p3.y

**4. Y 坐标的计算：**

最后，您根据 K 和 a 来计算 y 坐标：



### 2.3.2 断层模型网格化

## 2.4地层模型构建

### 2.4.1 CDT剖分数据源提取

### 2.4.2 地层数据网格化

2.4.1 钻孔

2.4.2 剖面模型（勘探线、剖分的剖面）\*\*坐标转换等\*\*

2.4.3 地层模型

2.4.4 断层模型

2.4.5 .....

## 2.5 本章小结

2.4.1 地层层间距计算 \*\*高程不同颜色显示\*\*（\*\*有额外的计算？三级：二级）（公式）

{"mathml":"<math style=\"font-family:stix;font-size:16px;\" xmlns=\"http://www.w3.org/1998/Math/MathML\"><mstyle mathsize=\"16px\"><mi>d</mi><mo>=</mo><msqrt><msup><mfenced><mrow><msub><mi>x</mi><mn>2</mn></msub><mo>-</mo><msub><mi>x</mi><mn>1</mn></msub></mrow></mfenced><mn>2</mn></msup><mo>+</mo><msup><mfenced><mrow><msub><mi>y</mi><mn>2</mn></msub><mo>-</mo><msub><mi>y</mi><mn>1</mn></msub></mrow></mfenced><mn>2</mn></msup><mo>+</mo><msup><mfenced><mrow><msub><mi>z</mi><mn>2</mn></msub><mo>-</mo><msub><mi>z</mi><mn>1</mn></msub></mrow></mfenced><mn>2</mn></msup></msqrt></mstyle></math>","origin":"MathType for Microsoft Add-in"}

# 基于WebGL可视化方法 10

## 3.1 Web 可视化技术

1、WebGL 技术原理

WebGL（Web Graphics Library）是一种基于 OpenGL ES 2.0 的 JavaScript API，用于在浏览器中渲染高性能的 3D 图形。WebGL 允许开发者直接与 GPU 进行交互，利用浏览器的 Canvas 元素渲染复杂的三维场景。

WebGL 的核心是通过顶点着色器和片元着色器来控制图形渲染流水线，实现复杂的三维模型绘制。它支持直接在浏览器中加载、解析和渲染 3D 模型，并与 HTML5 的其他技术（如音频、视频）无缝集成，适用于游戏、数据可视化、科学计算等领域。

**主要内容**：

* WebGL 的架构与工作流程。
* 着色器（Shader）的编写与使用。
* 与 GPU 的通信机制。
* WebGL 在三维模型渲染中的优势与局限性。

2、 Three.js 实现技术

Three.js 是基于 WebGL 的 JavaScript 3D 图形库，它封装了 WebGL 的底层 API，使开发者能够更加方便地构建复杂的三维场景、物体和动画效果。Three.js 提供了大量的几何体、材质、灯光、相机等工具，简化了三维可视化的开发过程。

Three.js 的优势在于它降低了 WebGL 的使用门槛，并且具有很强的扩展性，可以处理模型加载、场景管理、动画和交互等任务。

**主要内容**：

* Three.js 的架构与组件介绍。
* 使用 Three.js 构建三维场景的基本步骤。
* 几何体、材质和灯光的应用。
* Three.js 与 WebGL 的对比与优劣分析。

## 3.2构建三维可视化场景（公式）

在三维可视化场景的构建过程中，需要定义场景中的对象、光源、相机和渲染方式。基于几何学和图形学的基本原理，通过数学公式定义模型的变换、投影和光照效果。

场景构建的主要步骤：

1、定义三维坐标系：所有对象的坐标系需要统一到一个世界坐标系中。

2、对象的几何构造：通过几何体（如立方体、球体等）或者导入的 3D 模型文件（如 OBJ）构建三维对象。

3、视图变换公式：物体从模型坐标系转换到世界坐标系、相机坐标系以及投影坐标系的过程，可以通过矩阵变换公式表示。

模型变换矩阵： 

视图变换矩阵：

投影矩阵（透视投影）：

公式推导：

-通过三维变换矩阵的公式推导，实现模型的平移、旋转和缩放。

-讨论几何体的法向量计算和光照模型（如 Phong 模型、Blinn-Phong 模型）的公式。

## 3.3模型渲染方法

### 3.3.1 模型纹理uv计算方法（公式）方法的创新点

1. 盒子包围盒的长宽高计算

X 轴方向的长度：

dx = max.x - min.x

Y 轴方向的长度：

dy = max.y - min.y

Z 轴方向的长度：

dz = max.z - min.z

2. 根据法向量的最大分量选择 UV 投影平面

法向量的每个分量的绝对值用于确定哪个分量最大，以此选择相应的 UV 投影平面：

如果|nX| >= |nY| 且 |nX| >= |nZ|，则选择 x 轴为主导分量：

- 投影平面为 YZ 平面。

如果 |nY| >= |nX|且 |nY| >= |nZ|，则选择 y 轴为主导分量：

- 投影平面为 XZ 平面。

如果|nZ| >= |nX|且|nZ| >= |nY|，则选择 z 轴为主导分量：

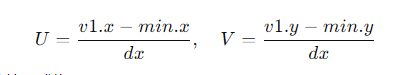
- 投影平面为 XY 平面。

3. UV 坐标的计算公式

UV 坐标的计算基于每个顶点的坐标以及包围盒的尺寸。

- 如果 Z 轴分量最大（type = 2），投影到 XY 平面：

- 如果 dx > dy，UV 坐标计算公式为：



- 否则，UV 坐标计算公式为：



如果 X 轴分量最大（type = 0），投影到 YZ 平面：

- 如果dz > dy，UV 坐标计算公式为：



- 否则，UV 坐标计算公式为：



如果 Y 轴分量最大（type = 1），投影到 XZ 平面

- 如果dx > dz，UV 坐标计算公式为：



- 否则，UV 坐标计算公式为：



## - 3.4交互功能（和3.3是否需要交换）

### 3.4.1 射线追踪

### 3.4.2 坐标转换

### 3.4.1 巷道漫游

### 3.4.2 点选功能

1. 三维可视化系统开发

（部分伪代码）

4.1系统架构设计

- 4.1.1 体系结构（原生H5开发\*\*系统升级优化\*\*为VUE3的框架开发）

- 4.1.2 技术流程

4.2系统功能实现（伪代码）

- 4.2.1

- 4.2.1

- 系统功能测试

\*\*创新点：系统更新换代 优化\*\* 提高开发效率、项目运行流畅等。

代码复用 （数据结构 可视化方式 api等等）

1. 应用实例
2. 结论与展望

- 6.1 结论

- 6.2 展望