**封面**

**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 国外研究现状 3

1.2.2 国内研究现状 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 断层模型构建 3](#_Toc152253751)

2.2.1 相交断层模型的构建方法 3

2.2.2 断层模型网格化 3

[2.3 地层模型构建 3](#_Toc152253751)

2.3.1 CDT剖分数据源提取 3

2.3.2 地层数据网格化 3

[2.4 地质模型集成 3](#_Toc152253751)

2.4.1 钻孔 3

2.4.2 剖面模型（勘探线、剖分的剖面） 3

2.4.2 地层模型 3

2.4.2 断层模型 3

[2.5 空间几何信息的计算 3](#_Toc152253751)

2.5.1 地层层间距计算 3

2.5.2（） 3

[**3 三维模型可视化方法** 5](#_Toc152253752)

[3.1 Web可视化技术 5](#_Toc152253753)

3.1.1 WebGL技术 3

3.1.2 Threejs技术 3

[3.2 构建三维可视化场景 5](#_Toc152253754)

[3.3 模型渲染方法 5](#_Toc152253754)

3.3.1 **巷道纹理uv计算方法** 3

3.3.2 巷道模型可视化 3

[3.4 三维地质模型交互功能 5](#_Toc152253754)

3.4.1 射线追踪 3

3.4.2 坐标转换 3

3.4.3 巷道漫游 3

3.4.4 点选功能 3

[**4 三维可视化系统开发** 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)

4.2.1 射线追踪 3

4.2.2 系统功能测试 3

[**5 应用实例** 8](#_Toc152253759)

[5.1 平台总体框架搭建 2](#_Toc152253750)

[5.2 构建三维场景以及三维地质模型集成 3](#_Toc152253751)

[5.3 三维地质模型交互功能实现 3](#_Toc152253751)

[5.4 三维地质模型信息集成与展示 2](#_Toc152253750)

[5.5 (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
2. 三维模型构建

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）

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;

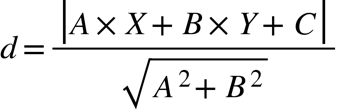
double B = X1 - X2;

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)

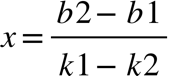
double y = x \* line1.k + line1.b

-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

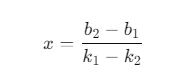
double x = (line2.b - line1.b) / (line1.k - line2.k);

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地层模型构建

2.3.1 CDT剖分数据源提取（\*\*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"}

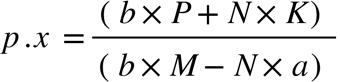
**2. Z 坐标的计算：**

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

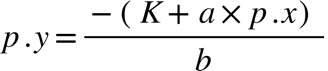
• K = c \* p.z + d

• 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 地层数据网格化

2.3.3 模型构建

2.4地质模型集成

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 空间几何信息的计算

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"}

2.4.2

1. 三维模型可视化技术与方法10

3.1 Web 可视化技术概述

3.1.1 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。

3.1.2 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）

**3.2 三维可视化场景的构建**

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 巷道纹理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：

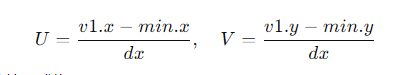
- 投影平面为 XY 平面。

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地质模型可视化（和3.3是否需要交换）

- 3.4.1 光照

- 3.4.2

3.5三维地质模型交互功能（公式）

- 4.2.1 射线追踪

- 4.2.2 坐标转换

- 4.3.1 巷道漫游

- 4.3.2 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