

关键要点

- 研究表明，`main.bin` 文件可能通过 `self.loadGeometries` 加载为二进制几何数据。
 - 看起来 `main.json` 文件通过 `instance.loadScene` 加载，并使用 `main.bin` 的几何数据映射到建筑模型。
 - 证据显示，建筑模型在场景中通过 "`blocks`" 对象提取并处理。
-

加载和映射过程

`main.bin` 的加载和解码

`main.bin` 文件似乎是通过 `self.loadGeometries` 函数加载的，该函数将几何体名称（如 "`main`"）转换为 "`main.bin`"，然后使用自定义的 `THREE.XHRLoader (Connection)` 加载二进制数据。加载后，二进制数据在 `parseBinaryGeometries` 函数中解码为 `Three.js` 的几何体对象。

`main.json` 的加载

`main.json` 文件看起来是通过 `instance.loadScene` 函数加载的，路径为 "`assets/scenes/main.json`"。加载时，会检查是否存在对应的几何体数据（从 `main.bin` 加载），并将其设置为场景加载器的缓冲区。

映射到建筑模型

加载的 `main.json` 包含一个名为 "`blocks`" 的对象，里面多个建筑模型 (`mesh`)。这些模型引用了 `main.bin` 的几何数据。在 `start` 方法中，建筑模型通过 `_getObjectByName("blocks").children` 提取，并传递给 `SVGTable` 类，用于生成场景的动态块。

详细分析报告

以下是基于 JavaScript 文件 (`main.nice.js`) 内容的详细分析，探讨文件如何加载 `main.bin` 文件并解码，以及如何加载 `main.json` 文件并将其映射到不同的建筑模型中。分析基于文件内容，并结合 `Three.js` 的常见用法进行推断。

背景与概述

`main.nice.js` 是一个基于 `Three.js` 的 3D 渲染应用程序，专注于处理复杂的 3D 场景，包括街道、建筑物、道路、交叉路口、汽车和云等。文件包含多个模块，每个模块负责不同的功能，如资产加载、场景构建和用户交互处理。根据您的描述，场景中的所有几何数据都是通过加载 `main.bin` 文件加载到内存中，然后通过 `main.json` 文件解码为不同的 `mesh` 数据，并最终映射到不同的建筑模型中。

1. `main.bin` 文件的加载和解码

- 加载机制：

- 文件中使用了自定义的几何体加载器 (`self.loadGeometries`, 位于模块 15)，负责加载二进制几何体文件 (如 `.bin` 文件)。
- 具体来说，`self.loadGeometries` 函数接受一个几何体名称列表，将其转换为带有 `.bin` 后缀的文件名 (例如 `main` 转换为 `main.bin`)，然后使用 `THREE.XHRLoader` 的一个自定义版本 (`Connection`, 位于模块 10) 来加载这些二进制文件。
- 代码片段：

```
javascript

self.loadGeometries = function(e, options) {
    e = _.map(e, function(canCreateDiscussions) { return canCreateDiscuss
    return fn(e, options || self.geometryPath, c);
};
```

- 这里，`e` 是几何体名称列表 (例如 `["main"]`)，`fn` 是加载函数，`c` 是自定义的加载器实例。
- 解码机制：

- 加载的二进制数据（`main.bin`）随后在模块 16 的 `parseBinaryGeometries` 函数中进行解析。
 - 虽然具体的解析代码未在提供的分析结果中显示，但从 `Three.js` 的常见用法来看，二进制几何体数据通常包含顶点、索引和其他几何属性，这些数据被解析为 `Three.js` 的 `BufferGeometry` 对象。
 - 例如，`Three.js` 的 `GLTFLoader` 也会处理类似的情况，其中 `.bin` 文件包含几何体数据，而 `.gltf` 或 `.json` 文件包含场景结构。
- **总结：**
- `main.bin` 文件通过 `self.loadGeometries` 加载，文件名被自动转换为 `main.bin`。
 - 加载的二进制数据在 `parseBinaryGeometries` 中解码为可用的几何体对象。

2. `main.json` 文件的加载和映射

- **加载机制：**
- 文件中使用 `instance.loadScene` 函数（位于模块 17）来加载场景文件，包括 `main.json`。
 - 具体来说，`instance.loadScene` 函数接受场景名称（如 `"main"`）、基础路径（如 `"assets/scenes/"`）和选项作为参数，加载对应的 JSON 文件（例如 `"assets/scenes/main.json"`）。
- **代码片段：**

```
javascript X ⌂ ▶ ⌂

instance.loadScene = function(name, data, options, callback) {
    return new _getServer(function(_emscripten_bind_Vector____destroy____0,
        var addedRenderer = shape.getGeometry(name);
        if (addedRenderer) {
            shape.sceneLoader.setBinaryGeometryBuffer(addedRenderer);
        }
        shape.loadScene(data + name + (callback || ".json")).spread(function
    });
};
```

- 这里，`name` 是场景名称（如 `"main"`），`data` 是基础路径（如 `"assets/scenes/"`），`callback` 默认为 `".json"`，因此加载的文件为 `"assets/scenes/main.json"`。

- **几何体数据的关联：**

- 在加载场景之前，代码会检查是否存在与场景名称对应的几何体数据（通过 `shape.getGeometry(name)`）。
- 如果存在（例如从 `main.bin` 加载的几何体），则将这些几何体数据设置为场景加载器的二进制几何体缓冲区（`(shape.sceneLoader.setBinaryGeometryBuffer(addedRenderer))`）。
- 这意味着 `main.json` 文件中的场景描述可以引用 `main.bin` 中加载的几何体数据。

- **映射到建筑模型：**

- 加载的场景（`main.json`）包含一个名为 `"blocks"` 的对象，该对象包含多个建筑模型（`mesh`）。
- 在 `_` 类的 `start` 方法中，代码通过 `_.getObjectByName("blocks").children` 提取这些建筑模型，并传递给 `SVGTable` 类进行进一步处理。
- 代码片段：

```
javascript
_.inherit(ref, {
  start: function(_) {
    var root_width = _.getObjectByName("blocks").children; // 建筑模型
    // 其他对象如道路、交叉路口等
    this.table = new SVGTable(root_width, root_height, table_options,
    }
});

```

- 这里，`_.getObjectByName("blocks").children` 返回 `"blocks"` 对象的所有子对象，这些子对象就是建筑模型。

- `SVGTable` 类（位于模块 48）负责将这些建筑模型组织成可重用的块（chunks），以实现场景的无限循环显示。

· 总结：

- `main.json` 文件通过 `instance.loadScene` 加载，文件路径为 `"assets/scenes/main.json"`。
 - 加载的 JSON 文件包含场景结构，包括 `"blocks"` 对象，该对象包含多个建筑模型。
 - 这些建筑模型引用了从 `main.bin` 加载的几何体数据。
 - 建筑模型随后被提取并用于构建场景的动态块。
-

3. 加载和映射的整体流程

以下是 `main.bin` 和 `main.json` 的加载及映射的完整流程：

1 加载几何体数据：

- 使用 `self.loadGeometries` 加载 `main.bin` 文件（文件名通过 `"main" + ".bin"` 生成）。
- 二进制数据在 `parseBinaryGeometries` 中解码为 `Three.js` 的几何体对象。

2 加载场景数据：

- 使用 `instance.loadScene("main", "assets/scenes/", options)` 加载 `main.json` 文件。
- 在加载场景之前，检查是否存在与 `"main"` 对应的几何体数据（从 `main.bin` 加载）。
- 如果存在，将几何体数据设置为场景加载器的缓冲区。

3 解析场景：

- `main.json` 文件被解析为 `Three.js` 的场景对象。
- 场景对象包含一个名为 `"blocks"` 的对象，该对象包含多个建筑模型（mesh）。

4 映射建筑模型：

- 在 `_` 类的 `start` 方法中，提取 `"blocks"` 对象的子对象（建筑模型）。

- 这些建筑模型被传递给 `SVGTable` 类，用于生成场景的动态块。

5 无限循环显示：

- `SVGTable` 类使用模块化算术（如 `i = i % TABLE_SIZE`）和动态加载机制，将建筑模型组织成可重用的块，实现场景的无限循环。
-

结论

· 加载：

- `main.bin` 通过 `self.loadGeometries` 加载，文件名转换为 "`main.bin`"。
- `main.json` 通过 `instance.loadScene` 加载，文件路径为 "`assets/scenes/main.json`"。

· 解码和映射：

- `main.bin` 的二进制数据在 `parseBinaryGeometries` 中解码为几何体对象。
- `main.json` 包含场景结构，包括 "`blocks`" 对象，该对象包含多个建筑模型。
- 建筑模型引用了从 `main.bin` 加载的几何体数据。
- 这些模型被提取并用于构建场景的动态块，实现无限循环显示。

· 关键函数和模块：

- `self.loadGeometries`：加载二进制几何体文件。
- `instance.loadScene`：加载 JSON 场景文件。
- `parseBinaryGeometries`：解码二进制几何体数据。
- `_` 类的 `start` 方法：提取建筑模型。

- `SVGTable` 类：处理建筑模型并生成动态块。
-

关键引文

- How to properly load a Json File in Threejs
- Loading scene from json - Questions - three.js forum
- The best way to save and open three.js scene - Questions - three.js forum
- learning-threejs/chapter-08/04-load-save-json-scene.html at master · josdirksen/learning-threejs
- Three.js load json string
- Best Practice for save/ load three js scenes - Questions - three.js forum
- Problems with load scene via html - Questions - three.js forum
- How can I save and then load a Three.js scene
- learning-threejs/chapter-08/03-load-save-json-object.html at master · josdirksen/learning-threejs
- JSON Object Scene format 4 · mrdoob/three.js Wiki · GitHub
- How do I load a gltf file that has multiple bin and texture files - Questions - three.js forum
- Loading gltf, bin and png from the backend - Questions - three.js forum
- GLTF loader. specify different paths for the gltf and bin files - Questions - three.js forum

- How to load raw gltf file with THREE.js
- Load a 3D GLB / GLTF model with Three.js - GLTFLoader - Three.js University
- Three.js Loading a .GLTF File
- Three.JS how can load bin GLB(GLTF) from string variable with GLTFLoader?
- Threejs
- Load 3D Models in glTF Format | Discover three.js
- webpack - Failed to load buffer (.bin) - Three.JS - Stack Overflow