

关键要点

- 研究表明，无限城市通过分块系统实现，相机移动时动态加载和重新定位块。
 - 无限循环通过模块化算术（如 `i = i % TABLE_SIZE`）实现，确保块在网格边缘循环。
 - 主要代码在模块 43（Scene）、48（SVGTable）、55（BaseObject）等实现。
-

直接回答

无限城市的实现是通过将城市分成多个块（chunks），当相机移动时，系统会动态加载或重新定位这些块，创造出无限延伸的幻觉。以下是具体机制：

相机移动与块更新

- 当您移动相机，系统会跟踪相机的位置（通过 `gridCoords`），并根据移动距离更新可见块。
- 例如，如果您向右移动，系统会加载新的块到右侧，同时将左侧的块重新定位到右侧，保持城市的连续性。

无限循环的实现

- 为了让城市无限循环，系统使用模块化算术（比如 `i = i % TABLE_SIZE`），确保当相机到达网格边缘时，块会“循环”回另一侧。
- 这就像在一个环形跑道上跑步，无论跑多远，您总会回到起点。

动态对象的管理

- 城市中的动态对象（如汽车和云）也会根据块的循环特性调整位置，确保它们在块之间无缝移动。

主要代码部分包括：

- **模块 43（Scene）**：处理相机移动和块更新。
- **模块 48（SVGTable）**：生成和循环块数据，使用模块化算术实现无限循环。
- **模块 55（BaseObject）**：管理动态对象的定位和环绕。

这些机制共同确保了城市的无限延伸，适合用户自由探索。

详细分析报告：JS 文件中无限城市的实现机制

背景与概述

您提供的 JS 文件（main.nice.js）是一个基于 Three.js 的 3D 渲染应用程序，专注于创建一个无限循环的城市场景。用户可以在其中自由移动相机，而城市会无限延伸，创造出一种无边界的幻觉。本报告将详细分析该文件如何实现这一功能，重点关注城市的无限循环机制以及相机移动时如何动态更新场景。

1. 无限城市的实现原理

无限城市的实现基于以下核心概念：

- **分块（Chunk）系统**：城市被分割成一个网格，每个网格单元称为“块”（chunk）。每个块包含城市的部分内容，如建筑、道路、交叉路口、汽车和云。
 - **动态加载与重新定位**：当相机移动时，系统会根据相机的位置动态加载或重新定位块，确保用户始终看到一个连续的城市场景。
 - **模块化算术（Modulo）**：通过在块坐标上使用模块化算术（如 `i = i % TABLE_SIZE`），实现了当相机到达网格边缘时，块会“循环”回另一侧，创造出无限平面的效果。
 - **对象环绕**：城市中的动态对象（如汽车和云）也会根据块的循环特性进行位置调整，确保它们在块边界处无缝转移。
-

2. 相机移动时的无限循环机制

当相机移动时，系统通过以下步骤实现城市的无限循环：

- **相机位置跟踪**：
 - 相机的位置通过 `gridCoords`（一个 `THREE.Vector2` 对象）来跟踪，该对象表示相机在块网格中的坐标。
 - 当用户通过拖动或平移操作移动相机时，`gridCoords` 的值会根据移动距离更新（例如 `this.gridCoords.x += eastPx` 和 `this.gridCoords.v += vertSpeed`）。

- 这一机制由模块 43 (Scene) 中的 `controls.on("move", ...)` 实现，监听用户的输入（如平移）并更新 `gridCoords`。

- 块更新：

- 系统调用 `refreshChunkScene` 方法（位于模块 43， `Scene` 类），根据更新后的 `gridCoords` 计算出当前可见块的坐标。
- 对于每个可见块，系统使用 `SVGTable` 类的 `getChunkData` 方法（模块 48）获取或生成块的数据。
- `getChunkData` 方法使用模块化算术确保块坐标在网格边缘时循环：

```
javascript
```

```
i = i % options.TABLE_SIZE, x = x % options.TABLE_SIZE;  
i < 0 && (i = options.TABLE_SIZE + i), x < 0 && (x = options.TABLE_SIZE +
```

- 这意味着如果块坐标超出网格大小（`TABLE_SIZE`），它会循环回网格的另一侧。例如，如果 `TABLE_SIZE` 为 9，当 `i = 10` 时，`i % 9 = 1`，从而回到网格的开头。
- 这种机制确保了城市的无限延伸，类似于在一个环形跑道上跑步，无论跑多远，您总会回到起点。

- 块重新定位：

- 如果相机移动到一个新的块区域，系统会移除当前不可见的块，并加载或重新定位新的块。
- 块的位置通过 `THREE.Object3D` 的 `position` 属性设置，确保它们在场景中正确对齐。
- 这一过程由 `refreshChunkScene` 方法实现，通过遍历 `chunkScene` 中的每个块，移除旧的块并添加新的块：

```
javascript

refreshChunkScene: function() {
    this.chunkScene.forEachChunk(function(results, columnGap, a) {
        var body = this.gridCoords.x + columnGap;
        var val = this.gridCoords.y + a;
        var v = this.table.getChunkData(body, val);
        results.remove(results.getObjectByName("chunk"));
        results.add(v.node);
    }.bind(this));
}
```

- 这里，`gridCoords.x` 和 `gridCoords.y` 与块的相对位置 (`columnGap`, `a`) 相加，

计算出当前可见块的坐标。

- 动态对象环绕：

- 城市中的动态对象（如汽车和云）由 `BaseObject` 类（模块 55）管理。
- `_updateTablePosition` 方法使用 `THREE.Math.euclideanModulo` 确保对象的位置在块边界处环绕：

```
javascript
```

```
var i = Math.floor(clamp(this.tablePosition.x + 40, MIN_BUFFER_ROWS) / rect.CHUNK_SIZE);
var name = Math.floor(clamp(this.tablePosition.z + 40, MIN_BUFFER_ROWS) / rect.CHUNK_SIZE);
```

- 如果对象超出当前块的边界，它会被重新附加到新块，并调整位置：

```
javascript
```

```
var min_x = clamp(this.position.x + 40, rect.CHUNK_SIZE) - 40;
var _depth = clamp(this.position.z + 40, rect.CHUNK_SIZE) - 40;
this.position.x = min_x;
this.position.z = _depth;
```

- 这一机制确保动态对象在块之间无缝移动，保持城市的连续性

3. JS 文件中实现无限循环的关键部分

以下是 JS 文件中实现无限循环的主要代码部分，整理为表格以便清晰展示：

模块编

号	类/功能	主要功能
50	config.js	定义网格大小（如 TABLE_SIZE 为 9）和块大小（如 CHUNK_SIZE 为 60），决定城市结构。
47	GlitchTextLetter	初始化 9x9 块网格，创建和遍历块，用于相机移动时的更新。
48	SVGTable	生成块数据，使用 getChunkData 实现模块化算术循环，确保无限延伸。
43	Scene	处理相机移动，更新 gridCoords，通过 refreshChunkScene 刷新可见块。
55	BaseObject	管理动态对象（如汽车）的定位，使用 euclideanModulo 确保对象在块间环绕。
59	Utils	提供辅助函数（如 getTablePosition），支持块管理和定位。

- 模块 50 (config.js)：

- 定义了关键常量，如 TABLE_SIZE（网格大小，9x9）和 CHUNK_COUNT（可见块数量，9），这些常量决定了城市网格的规模和结构。
- 例如，CHUNK_SIZE 为 60 单位，决定了每个块的物理大小。

- 模块 47 (GlitchTextLetter):

- `_initChunks` 方法初始化一个 9x9 的块网格，确保覆盖相机可能移动的范围。
- `_createChunkAt(x, time)` 方法创建位于特定网格坐标的块，并计算其中心位置 (`centeredX`, `centeredY`)，用于后续定位。
- `forEachChunk` 方法遍历所有块，用于在相机移动时更新它们。

- 模块 48 (SVGTable):

- `_generate` 方法初始化块网格 (`chunks`)，每个块包含随机生成的建筑、道路等，确保内容的多样性。
- `getChunkData(i, x)` 方法是实现无限循环的核心：
 - 使用模块化算术 (%) 将块坐标循环回网格内：

```
javascript
i = i % options.TABLE_SIZE, x = x % options.TABLE_SIZE;
i < 0 && (i = options.TABLE_SIZE + i), x < 0 && (x = options.TABLE_SIZE + x);
```

- 确保当相机移动到网格边缘时，块会无缝循环。例如，如果 `TABLE_SIZE` 为 9，当 `i = 10` 时，`i % 9 = 1`，从而回到网格的开头。
- `_getRandomChunk(x, y)` 方法生成或获取位于特定坐标的块，确保内容的多样性。

- 模块 43 (Scene):

- controls.on("move", function(eastPx, vertSpeed) {...}) 监听用户的平移操作，更新 gridCoords :

```
javascript
this.gridCoords.x += eastPx;
this.gridCoords.y += vertSpeed;
this.refreshChunkScene();
```

- refreshChunkScene() 方法根据新的 gridCoords 更新可见块：

- 遍历 chunkScene 中的每个块，移除旧的块并添加新的块：

```
javascript
refreshChunkScene: function() {
    this.chunkScene.forEachChunk(function(results, columnGap, a) {
        var body = this.gridCoords.x + columnGap;
        var val = this.gridCoords.y + a;
        var v = this.table.getChunkData(body, val);
        results.remove(results.getObjectByName("chunk"));
        results.addObject(v, "chunk");
    });
}
```

```
        results.add(v.node);
    }.bind(this));
}
```

- 这里，`gridCoords.x` 和 `gridCoords.y` 与块的相对位置 (`columnGap`, `a`) 相加，计算出当前可见块的坐标。

- **模块 55 (BaseObject):**

- `_updateTablePosition` 方法确保动态对象（如汽车）在块之间无缝移动：
 - 使用 `THREE.Math.euclideanModulo` 确保对象的位置在块边界处环绕：

```
javascript
```

```
var i = Math.floor(clamp(this.tablePosition.x + 40, MIN_BUFFER_ROWS) /  
var name = Math.floor(clamp(this.tablePosition.z + 40, MIN_BUFFER_ROWS)
```

- 如果对象超出当前块，它会被重新附加到新块，并调整位置：

```
javascript
```

```
var min_x = clamp(this.position.x + 40, rect.CHUNK_SIZE) - 40;  
var _depth = clamp(this.position.z + 40, rect.CHUNK_SIZE) - 40;  
this.position.x = min_x;  
this.position.z = _depth;
```

- **模块 59 (Utils):**

提供抽象函数如 `getTablePosition`, 用于处理垂直表格的网格坐标，又对块管理处理块的位置。

4. 总结

- **无限城市的实现:**

- 城市被分成一个 9×9 的块网格，每个块包含城市的部分内容。
- 当相机移动时，系统通过 `gridCoords` 跟踪相机的位置，并使用 `refreshChunkScene` 更新可见块。
- `getChunkData` 方法使用模块化算术确保块在网格边缘时循环，创造无限平面的效果。
- 动态对象（如汽车）通过 `_updateTablePosition` 方法在块之间无缝转移。

- **关键代码部分:**

- **配置:** 模块 50 (`config.js`)，定义网格和块大小。
- **块管理:** 模块 47 (`GlitchTextLetter`)，初始化和遍历块。
- **表格与数据:** 模块 48 (`SVGTable`)，生成和循环块数据。
- **相机控制:** 模块 43 (`Scene`)，处理相机移动和块更新。
- **对象环绕:** 模块 55 (`BaseObject`)，管理动态对象的定位。

- **核心机制:**

- **分块系统:** 将城市分割成可重用的块。

- **动态加载**: 根据相机位置加载或重新定位块。
- **模块化算术**: 确保块在网格边缘时循环。
- **对象管理**: 确保动态对象在块之间无缝转移。

通过以上机制，JS 文件成功实现了无限循环的城市场景，允许用户在其中自由探索，而无需担心边界限制。

关键引文

- 如何在 Three.js 中创建无尽世界
- 无限城市展示
- 使用 Three.js 编码无限 3D 世界
- 使用 Three.js 创建无限管道
- 使用 Three.js 创建无限地形