STELLARA 大作业文档

> 未央-软件21 贾泽林 姚汝昌 韩枢辰 伍圣贤

一、使用说明

1.1 项目运行

在本仓库的根目录下运行以下命令（需要预先安装node.js和npm）：

```
npm install
npx vite build
npx vite preview
```

在浏览器中访问提示的地址（一般是`http://localhost:4173/`）即可。

1.2 项目结构

```
.
├── README.md
├── index.html
├── stellara
│   ├── main.js
│   ├── core
│   │   ├── app.js
│   │   ├── celestial_object.js
│   │   └── constant.js
│   │   └── mesh.js
│   │   └── rotation.js
│   │   └── shader_patcher.js
│   │   └── solar_system_data.js
│   │   └── solar_system_objects.js
│   └── assets
│       ├── texture
│       │   ├── earth.jpg
│       │   ├── earth_map.jpg
│       │   ├── earth_water.jpg
│       │   ├── moon.jpg
│       │   ├── sun.jpg
│       ├── css
│       │   ├── style.css
```

二、实现思路

本项目基于Three.js实现，主要分为以下几个部分：

2.1 场景搭建

2.1.1 创建星体

编写类`CelestialObject`，用于创建星体对象。星体对象包括名称、贴图、轨道、自转、子星体等基本信息，以及若干方法用于更新星体状态。

2.1.2 贴图渲染

- mesh贴图
  本项目中所有星体（太阳、地球、月球）都实现了表面贴图，且地球有两种不同贴图可供切换。
- shader修改
  通过建立遮挡面积的数学模型并对Three.js库中预设的OpenGL shader代码进行修改，本项目实现了对日食与月食现象的精确演示。例如日食发生时，地球上的月影从中心向四周深度逐渐减淡，符合不同观察点处月球所实际遮挡的太阳面积。

2.1.3 星体运动

- 公转
  本项目参考开源项目Celestia，基于非常精确的太阳系主要星体轨道模型VSOP87计算不同时刻地球相对太阳的位置，并使用一个较好的近似算法计算月球相对地球的位置。为体现本项目地月公转模型的精确程度，只需设置到任一历史上日食发生的时刻，即可在本项目中观察到日食的发生，且月影在地球表面的轨迹符合实际。
- 自转
  本项目中所有星体均实现了自转，且自转轴方向与自转速度基于Wikipedia上的相关数据，符合事实。与地月自转相关的物理现象，如地轴倾角、昼夜变换、地月潮汐锁定等，可在本项目中直接观察到。

2.1.4 轨道绘制

通过采样星体在一个周期内的位置，绘制星体的轨道。轨道的绘制采用`LineBasicMaterial`。为优化性能，采样点采用队列实现，无需每帧重新计算。需要指出，由于星体运动是基于真实的天文数据，不存在严格意义的周期性，因此绘制的轨道**不是完全闭合的**，这与实际情况一致。

2.2 页面逻辑

2.2.1 渲染逻辑

本项目基于Three.js的渲染引擎，每一帧调用`app.animate()`方法，渲染整个场景。在本项目中，我们在同一场景下使用两个相机和两个渲染器，分别渲染主视角和星体视角，后者用于展示特殊天文现象（日食、月食等）。

- 主视角
  主视角用于展示整个太阳系的运行情况，包括各个行星的运行轨迹、自转等。用户可以切换视野中心并控制缩放，基于`OrbitControls`实现。用户可以通过按钮或直接点击，切换视野中心的星体。
- 星体视角
  本部分用于展示日食、月食等特殊天文现象。此视角仅在特定条件下展示，如用户点击日食按钮，会模拟地球表面观测日食的情况；用户点击月食按钮，会模拟月球表面观测的视角。

2.2.2 交互控制

本项目使用事件总线 (event bus) 的设计模式实现组件之间的通信。例如，页面上的各个按钮是事件的发布者 (publisher)，计算模型则为事件的订阅者 (subscriber)。当某个按钮被点击时，相应的事件被发布，事件总线调用模型所注册的回调函数，完成对事件的响应。在这一模式中，各个组件不存在直接的依赖关系，从而降低了代码的耦合度。

2.2.3 界面设计

本项目的界面设计采用简洁明了的HTML结构和CSS样式，以实现良好的用户体验和易于操作的界面。界面主要包括时间设置、视角切换、视图切换、地球换肤、天文现象展示、时间速度控制等功能。其中关于时间设置部分，利用了Flatpickr（一个轻量级且功能强大的 JavaScript 日期选择器库）。关于日食月食等天文现象的展示，实现了在右下角显示一个小窗口用于展示太阳与地球（月球）的相对位置关系，在日食月食结束时窗口消失，注意，此处采用的是同一场景下的多相机渲染（相当于展示的是实际动画渲染情况，而非特殊绘制日食月食情况）。由于渲染的太阳、地球、月亮关系与实际情况完全相符，所以展示的日食月食现象也是实际发生的，相关时间数据来源于Wikipedia。

页面布局

页面采用居中布局，包含多个交互组件。布局结构如下：

- 顶部包括时间设置按钮、日期时间选择器、视角切换、视图切换、地球换肤按钮。
- 中间部分为天文现象展示的下拉菜单。
- 底部为时间速度控制按钮。
- 左下角为时间流速和实时时间。
- 右下角为一个模拟3D场景的容器。

样式设计

本项目的界面设计遵循简洁明了的原则，采用了易于操作的布局和样式，以确保用户体验的友好性和界面的直观性。主要样式设计如下：

- `body`：设置为无边距，以确保页面元素紧贴浏览器边缘，保持布局的完整性和一致性。
- `.dropdown`：下拉菜单的容器，设置为相对定位并行内块显示。通过`position: relative;`确保下拉内容的定位相对于此容器。
- `.dropdowncontent`：下拉内容的样式，初始设置为隐藏，通过`display: none;`，在用户悬停时显示。设置背景色、最小宽度、阴影等以提高视觉效果。
- `.dropdowncontent a`：下拉菜单项的样式，设置内边距、颜色、文本装饰等，以提供良好的用户交互体验。
- `.dropdowncontent a:hover`：下拉菜单项的悬停效果，背景色变亮，提升可读性和点击体验。
- `.dropdown:hover .dropdowncontent`：通过悬停`dropdown`容器，显示下拉内容。
- `#CenteredFlexColumn`：居中对齐的容器，使用Flexbox实现垂直居中排列，并设置适当的间距和对齐方式。容器的绝对定位和`transform`用于居中显示。
- `#pickerContainer`：日期时间选择器的容器，初始隐藏，悬停或点击显示。设置背景色、内边距等样式。
- `#showPickerButton:hover + #pickerContainer`：当鼠标悬停在显示日期选择器按钮上时，显示日期选择器容器。
- `#datetimePicker`：日期时间选择器的输入框，设置宽度和文本对齐方式。
- `.TimeSpeed`：时间速度控制区域，使用Flexbox水平居中排列按钮，并设置底部位置。
- `.TimeSpeed button`：时间速度控制按钮的样式，设置宽度和高度，确保按钮的统一视觉效果。
- `#ReturnSpeed`：特殊的返回速度按钮，设置宽度和位置，以适应按钮的排版需求。
- `.Astronomy`：天文现象展示区域，使用Flexbox水平居中排列下拉菜单，设置间距以分隔菜单项。

- `#nowtime` 和 `#timespeed`：分别显示当前时间和时间速度的区域，定位在页面左下角，使用白色文字以提高对比度和可读性。
- `#container`：3D场景的容器，设置绝对定位、背景色、边框和透明度。位置设置在页面右下角，大小适应内容显示需求。
- `.flatpickr-calendar`：日期选择器的样式，设置宽度与界面元素一致，以保证视觉统一。
- `button`：按钮的基本样式，设置高度、宽度、边框、圆角、背景色和字体大小。通过`:hover`和`:active`伪类定义按钮的悬停和点击效果。
- `#HalfSpeed`、`#TwotimesSpeed`、`#Stop`：时间速度控制按钮的特定样式，设置圆角，以实现视觉上的分隔效果。
- `.buttonline`：按钮行容器的样式，使用Flexbox布局，设置按钮间的间距，并防止换行。

这些样式和布局的设计确保了界面的整洁性、功能性和用户友好性，使得用户能够直观地进行操作和交互。

三、实现难点

3.1 贴图对齐

在设置地球贴图时需要调整贴图的对齐，使得地球的北极与贴图的北极对齐。这一过程需要对贴图进行几何变换操作，以使得地球的自转轴与贴图的自转轴对齐。具体而言，需要将原二维图计算球坐标中的位置，然后将球坐标转换为贴图坐标，最后在构建geometry时更正。

3.2 修改预设shader

为了展示日食与月食现象，必须计算场景中各物体的阴影。Three.js库中内置的shader使用shadow map技术，支持一般场景的阴影计算。但是，本项目的场景非常特殊：从太阳观察，地球和月球的角宽度非常小（$<10^{-4}rad$），这导致shadow map在可接受的分辨率下无法工作（经测试，shadow map的长宽即使均设为10000px，渲染出的阴影仍非常粗糙）。因为场景中所有的对象均为球体，在数学上容易描述，所以我们考虑适当修改库预设的shader代码，自行实现对日食与月食的精确绘制。
具体而言，在绘制每个星体时，我们先将场景中其他星体的位置通过uniform传入fragment shader。在计算当前像素受到的太阳光照强度时，额外检查是否有某个其他星体对太阳形成遮挡。如果有，则计算遮挡面积相对于太阳视表面积的比例，相应地衰减光照强度。
该过程的难点一在于遮挡比例的计算以及shader代码的编写，二在于探索如何修改库预设的shader。由于修改预设shader是一个相对少见的需求，相关资料稀缺，我们需要自行研究Three.js的文档和源码，解决修改过程中产生的与原有逻辑的冲突。我们的最终实现方案如下：利用库提供的一个shader编译前的回调onBeforeCompile修改shader的代码与uniform列表，并同时记录下shader本身。在每一帧渲染时，即可通过记录的shader更新相关的uniform。

3.3 多相机渲染

场景的渲染依赖`webGL`的渲染器实现，在同一场景下使用两个相机和两个渲染器，分别渲染。关键在于展示日食月食时，需要确定相机的位置和视角，以及如何切换相机。为模拟地表视角，我们通过星体坐标关系得到了大致的相机坐标。在主视角，我们通过`OrbitControls`实现了相机的控制，用户可以通过鼠标拖动、滚轮缩放等方式控制相机的位置，也可以直接点击星体或通过按钮切换视野中心。

四、分工情况

本项目采用git进行版本控制。仓库地址为：GitHub。
分工如下：

- 贾泽林：负责项目框架搭建、运动控制、场景搭建、渲染逻辑、文档编写
- 姚汝昌：负责贴图渲染、场景搭建、轨道控制、文档编写
- 韩枢辰：负责页面交互控制、日食月食展示、文档编写
- 伍圣贤：负责星体页面设计、文档编写

五、参考资料

- Three.js
- Three.js Examples
- WebGL Fundamentals
- Celestia
- NASA Orbit Viewer
- Planet Texture Maps
- flatpickr
- Wikipedia Solar Eclipse
- Wikipedia Lunar Eclipse