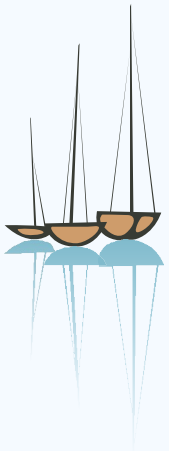
“精灵深林”课程设计报告

小组成员：刘洋、方纪陈

汇报人：刘洋

目录



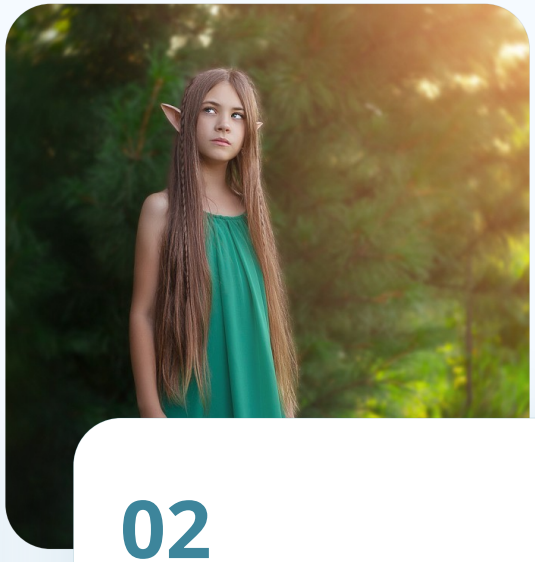
课程设计概述

选题背景与意义

01

小组内部想法与谈论

围绕“幻想”这一主题，我们谈论了许久，最后定下这个“精灵深林”



02

“精灵”与幻想

在我们看来“精灵”很好的符合了“幻想”这一主题

03

自身实力

“精灵”的范围及其广泛，所以它的上下限都很高，是我们能力所能及且提高能力的方案。

技术选型与工具使用

根据课程目标和学生基础，我们小组主要选用了 JavaScript 、 html 、以及 Css 等编程语言

主要在 HBuilder X 上编写代码同时也使用 Edge 浏览器的开发者工具与在 Edge 运用

以浏览器运行结果为参考，不断更新和修改课程设计

选择编程语言

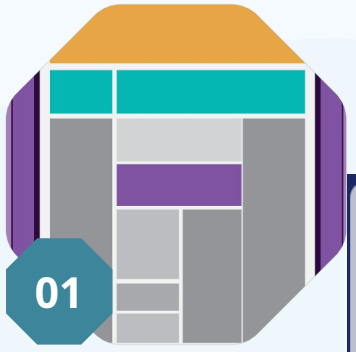
开发环境搭建

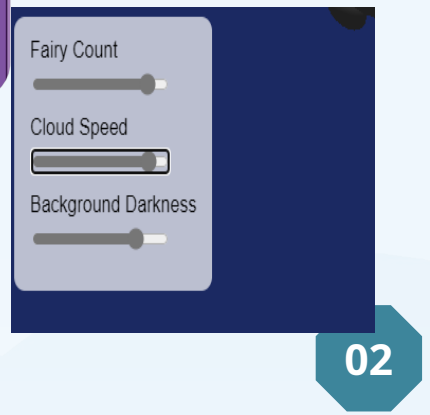
版本控制系统应用

交互控件与场景功能叙述

响应式布局设计

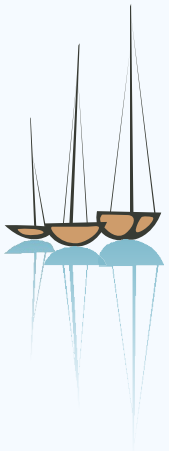
设计时考虑不同设备的显示效果，确保窗口在各种屏幕尺寸下均能自适应。





交互控件的动态调整

提供 3 个交互控件，实现用户对“精灵”数量、白云速度以及背景颜色的控制



场景元素建模

光照效果实现

01

环境光

创建了一个环境光，为整个场景提供一个基础的均匀光照效果，其颜色为白色（ 0xffffff ），强度为 0.6 。

```
ambientLight = new THREE.AmbientLight(0xffffff, 0.6);
scene.add(ambientLight);
```

02

平行光

创建了一个平行光，有特定的方向和位置（通过设置 position 属性），颜色同样为白色，强度为 0.8 ，用于模拟类似太阳光等有方向的光照

```
directionalLight = new THREE.DirectionalLight(0xffffff, 0.8);
directionalLight.position.set(10, 20, 10);
scene.add(directionalLight);
```

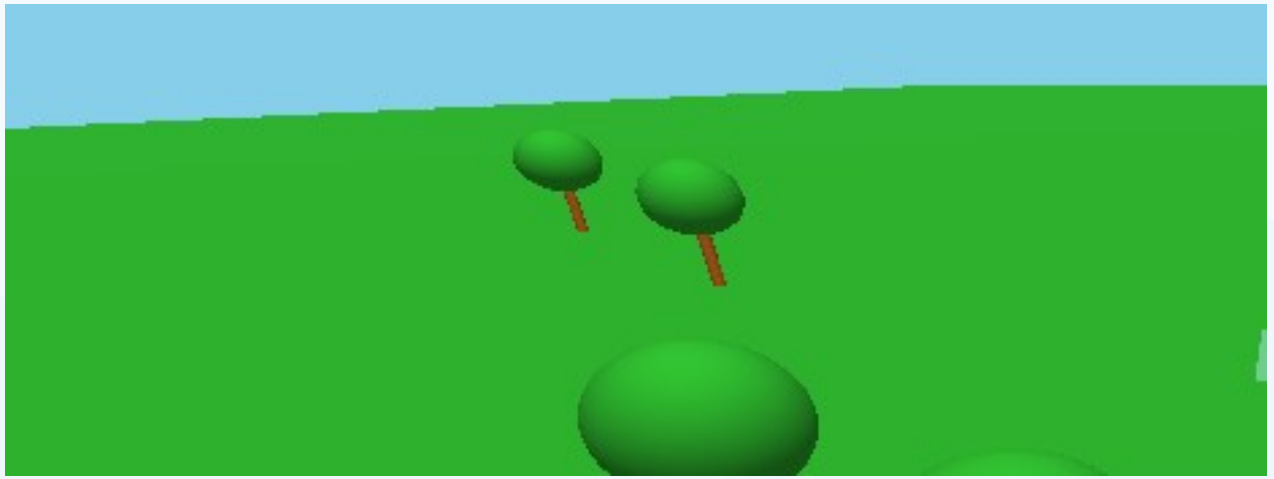
03

相机

创建了一个透视相机对象，用于定义观察三维场景的视角、位置等参数。通过设置相机的位置以及看向的目标点（ lookAt 方法），决定了场景呈现的视觉效果

```
camera = new THREE.PerspectiveCamera(75, window.innerWidth
camera.position.set(0, 10, 30);
camera.lookAt(0, 0, 0);
```

地面构建与材质选择



创建了一个平面几何体（ PlaneGeometry ）来表示地面，尺寸为 500×500 ，搭配一个标准材质（ MeshStandardMaterial ）设置颜色，然后通过 Mesh 将几何体和材质组合起来形成地面网格对象，并设置其旋转和位置等在三维空间中的姿态。

```
const groundGeometry = new THREE.PlaneGeometry(500, 500);
const groundMaterial = new THREE.MeshStandardMaterial({ color: 0x228B22 });
const groundMesh = new THREE.Mesh(groundGeometry, groundMaterial);
groundMesh.rotation.x = -Math.PI / 2;
groundMesh.position.y = -1;
```

采用绿色一方面与“深林”相符合一方面也让人感到舒适。

小精灵动画设计

”小精灵”使用了多个函数分别来构建他的身体、翅膀和光环。

身体：利用圆柱体几何体构建精灵的身体部分，配合相应材质组成网格对象。

```
const bodyGeometry = new THREE.CylinderGeometry(0.2, 0.3, 1, 32);
const bodyMaterial = new THREE.MeshStandardMaterial({ color: 0xffc0cb });
const bodyMesh = new THREE.Mesh(bodyGeometry, bodyMaterial);
```

头部：采用球体几何体创建精灵的头部，设置好材质以及在三维空间中的位置（相对其父对象，即精灵整体的位置偏移）。

```
const headGeometry = new THREE.SphereGeometry(0.3, 32, 32);
const headMaterial = new THREE.MeshStandardMaterial({ color: 0xffe4b5 });
const headMesh = new THREE.Mesh(headGeometry, headMaterial);
headMesh.position.y = 0.8;
```

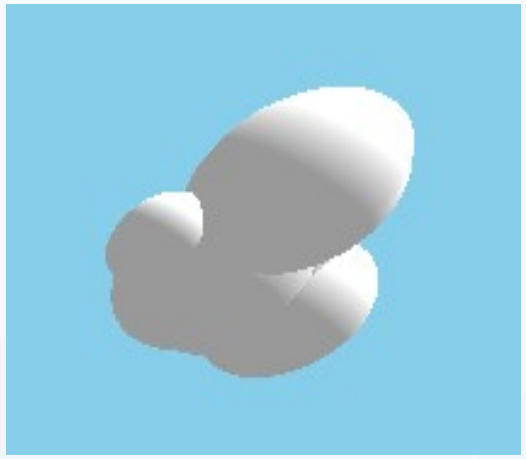
翅膀：通过平面几何体创建翅膀，设置材质、位置以及旋转角度等三维属性。

```
const wingGeometry = new THREE.PlaneGeometry(1, 1.5);
const wingMaterial = new THREE.MeshStandardMaterial({ color: 0xadd8e6, transparent: true, op
const leftWing = new THREE.Mesh(wingGeometry, wingMaterial);
const rightWing = new THREE.Mesh(wingGeometry, wingMaterial);
leftWing.position.set(-0.6, 0.3, 0);
rightWing.position.set(0.6, 0.3, 0);
leftWing.rotation.y = Math.PI / 6;
rightWing.rotation.y = -Math.PI / 6;
```

使用圆环几何体（ TorusGeometry ）来构建光环，配置材质并设置其在三维空间中的位置和旋转姿态，最后将这些组成部分组合成一个精灵整体（通过 THREE.Group ）添加到场景中。

```
const haloGeometry = new THREE.TorusGeometry(0.4, 0.05, 16, 100);
const haloMaterial = new THREE.MeshStandardMaterial({ color: 0xffff00, emissive: 0xffff00 })
const haloMesh = new THREE.Mesh(haloGeometry, haloMaterial);
haloMesh.position.y = 1.2;
haloMesh.rotation.x = Math.PI / 2;
```

白云构建



白云的构建，我们小组采用了较简单的做法：用多个球状几何体来模拟模拟云朵的各个组合部分

```
const cloudGeometry = new THREE.SphereGeometry(Math.random() * 2 + 1, 32, 32);
const cloudMaterial = new THREE.MeshStandardMaterial({ color: 0xffffff });
const cloudMesh = new THREE.Mesh(cloudGeometry, cloudMaterial);
cloudMesh.position.set(
    Math.random() * 5 - 2.5,
    Math.random() * 2 - 1,
    Math.random() * 5 - 2.5
);
```

为了使得”深林”更有趣，我们将白云移动起来了：

```
clouds.forEach(cloud => {
    cloud.position.x += cloud.userData.direction * cloudSpeed * 0.1;
    if (Math.abs(cloud.position.x) > 100) {
        cloud.userData.direction *= -1;
    }
});
```

根据云朵的移动方向和速度属性来更新其在三维空间中 x 轴方向的位置，并且当超出一定范围时改变移动方向，实现云朵飘动并循环的动画效果。最后通过 renderer.render(scene, camera) 不断渲染更新后的场景来展示动画变化。

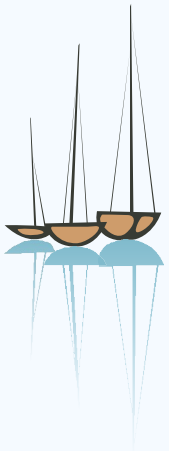
webGL 的运用

```
renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.getElementById('scene-container').appendChild(renderer.domElement);
```

document.getElementById('scene-container').appendChild(renderer.domElement);
这里通过 THREE.WebGLRenderer 创建了一个 WebGL 渲染器对象，它是 Three.js 库中基于 WebGL 技术用于在网页上渲染三维场景的核心组件。然后设置了渲染器的尺寸，使其能适配当前浏览器窗口的大小，并将渲染器生成的 DOM 元素（ renderer.domElement ）添加到页面中 id 为 scene-container 的 HTML 元素内，这样后续渲染出的三维场景才能在网页上显示出来。

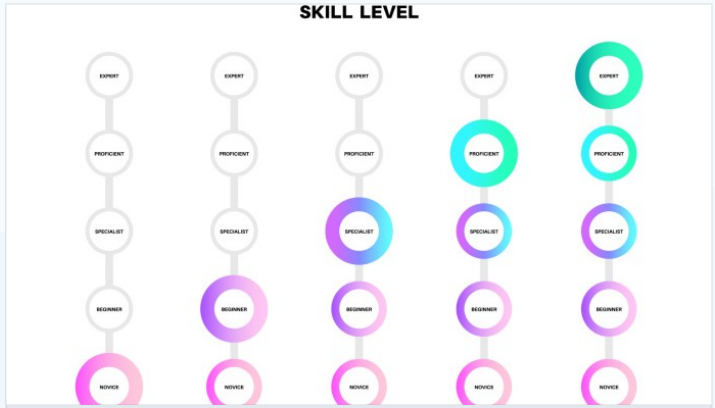
```
function animate() {
    requestAnimationFrame(animate);
    // 一系列更新场景中物体状态（如精灵、云朵的动画相关属性更新等）的代码
    renderer.render(scene, camera);
}
```

requestAnimationFrame 函数用于创建一个平滑的动画循环，它会在下一次浏览器重绘之前调用传入的函数（这里就是递归调用 animate 函数自身），保证动画以合适的帧率进行更新。而 renderer.render(scene, camera) 语句则是利用创建好的 WebGL 渲染器，将包含了各种三维物体的场景（ scene ）按照相机（ camera ）所设定的视角和投影等参数进行渲染，最终把三维场景绘制到页面上对应的 DOM 元素区域中，呈现出可视化的三维效果，这是 WebGL 在整个代码中用于实现渲染展示三维内容的核心操作。
总的来说，代码中通过 THREE.WebGLRenderer 来初始化渲染器，并结合 requestAnimationFrame 与 renderer.render 等操作来构建起基于 WebGL 的三维场景渲染流程，使得整个三维场景能够在网页环境中正常显示和动态更新。



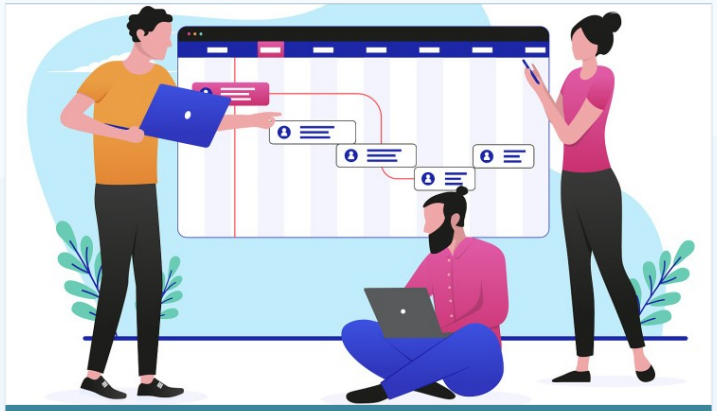
团队分工与合作

分工明细



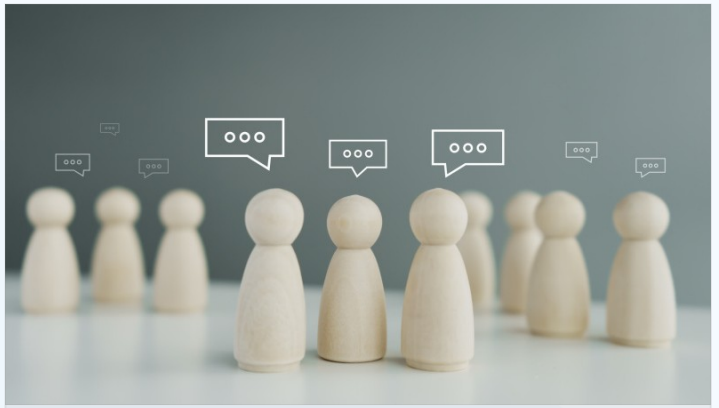
角色分配

刘洋给出整体规划与方向，给出所需实现内容与方向；方纪陈实现具体细节



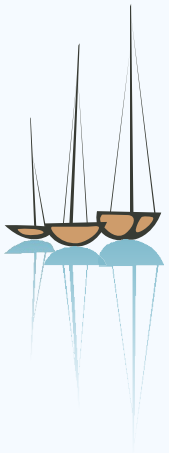
任务规划

各成员按照安排规划，有节奏推进任务



沟通协调机制

在完成自己任务的同时与对方积极交流实现进程的同步



第六章

自评与总结

回顾课程设计与收获

进度安排

1. **第一阶段**：小组内部讨论，确定大概方向、所用语言、技术等
2. **第二阶段**：开始行动，明确各成员间的分工，在完成自己任务同时与小组成员交流。
3. **第三阶段**：将各成员的成果结合，谈论与修改课程设计的效果与细节 。
4. **第四阶段**：整理项目代码、撰写详细课程设计报告，准备展示材料，确保项目可按提交要求完整呈现，按时提交源代码、文字材料、展示报告及相关附件到课程网站与指定个人网站展示。

本次课程设计收获还是很多的但也有不足之处，比如对整体进程的安排比较急促，以及最后作品在我们看来差强人意。但本次课程设计，让我们对 webgl 的认识得到进一步的加深与理解，同时在完成课程设计途中，我们也获得了许多的乐趣，在面对最终的成品后；虽然可能不是很精美但是让我们感觉到所付出的努力得到了回报。

谢谢

汇报人：刘洋