# 展示平台设计说明书

## 一.设计目的

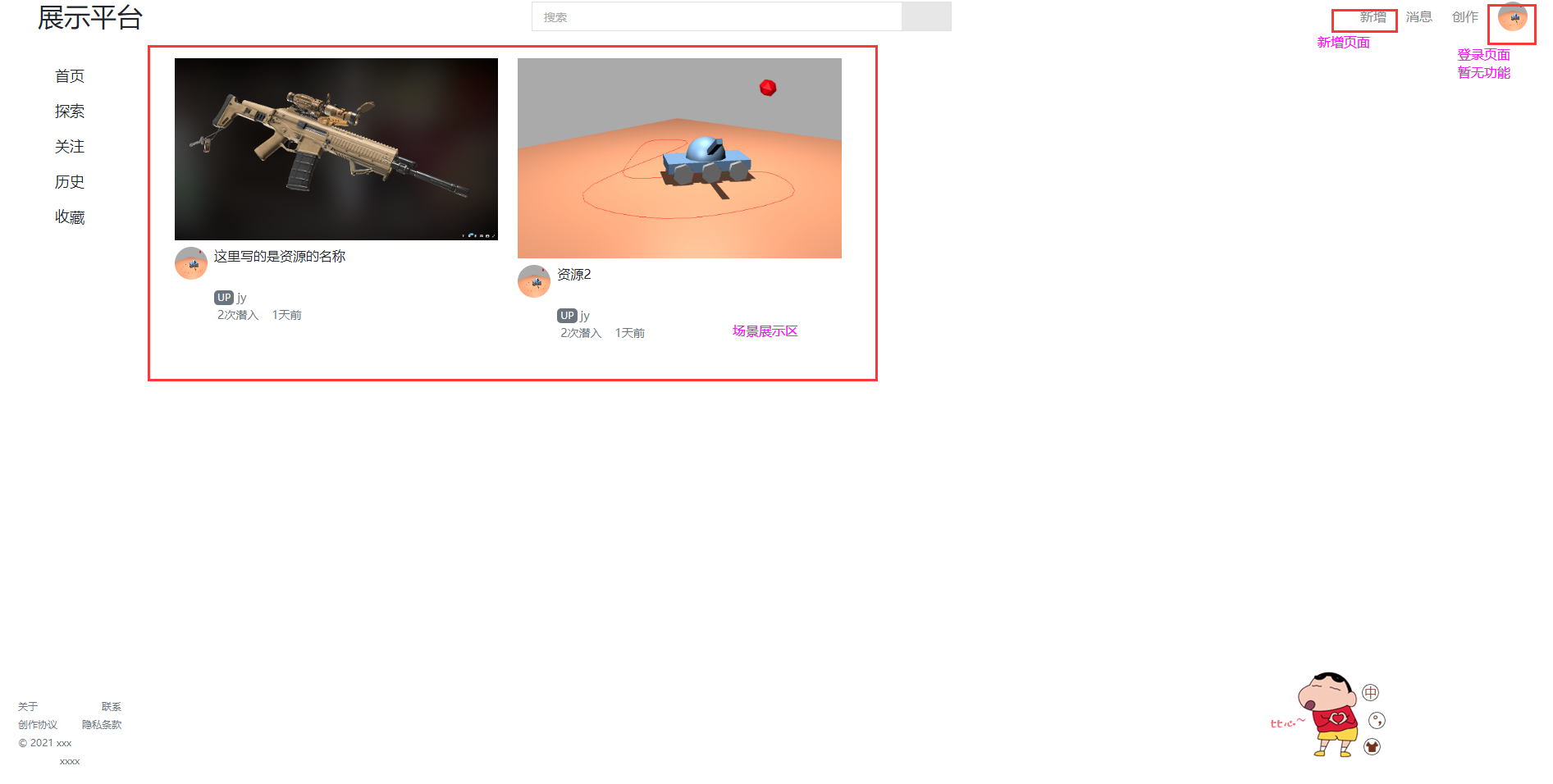
可载入任意包，场景展示（内容展示）平台，预计可支持three.js（webGL），gltf等3d模型组件，后续可支持将文件打包至此平台进行简单展示与分享。用户可以将自己所做的模型或者游戏场景发布到此平台进行展示。

## 二.功能设计

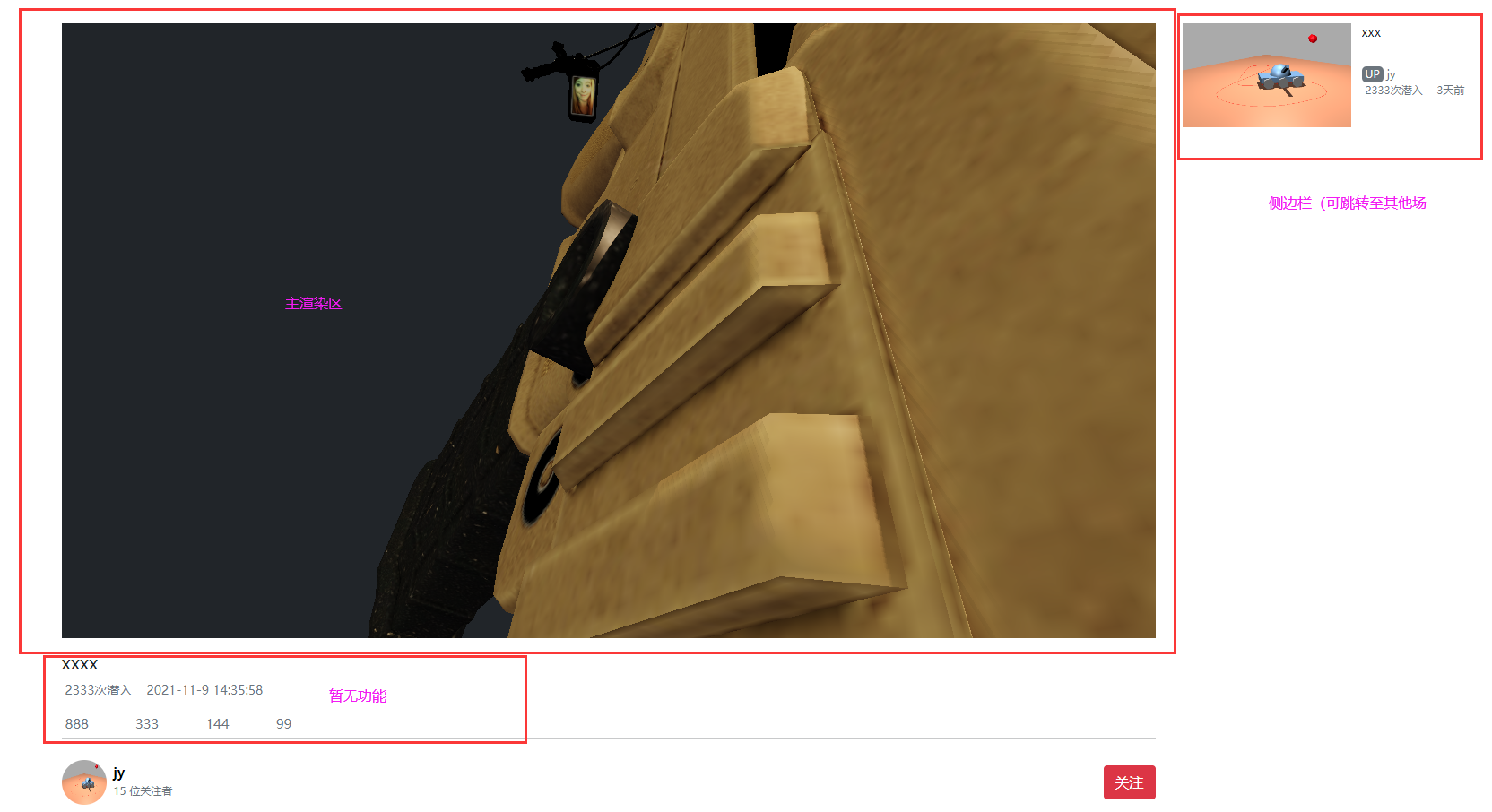
### 1.页面框架采用bootstrap模板：

初版设计原型图如下：

主页：



场景区：



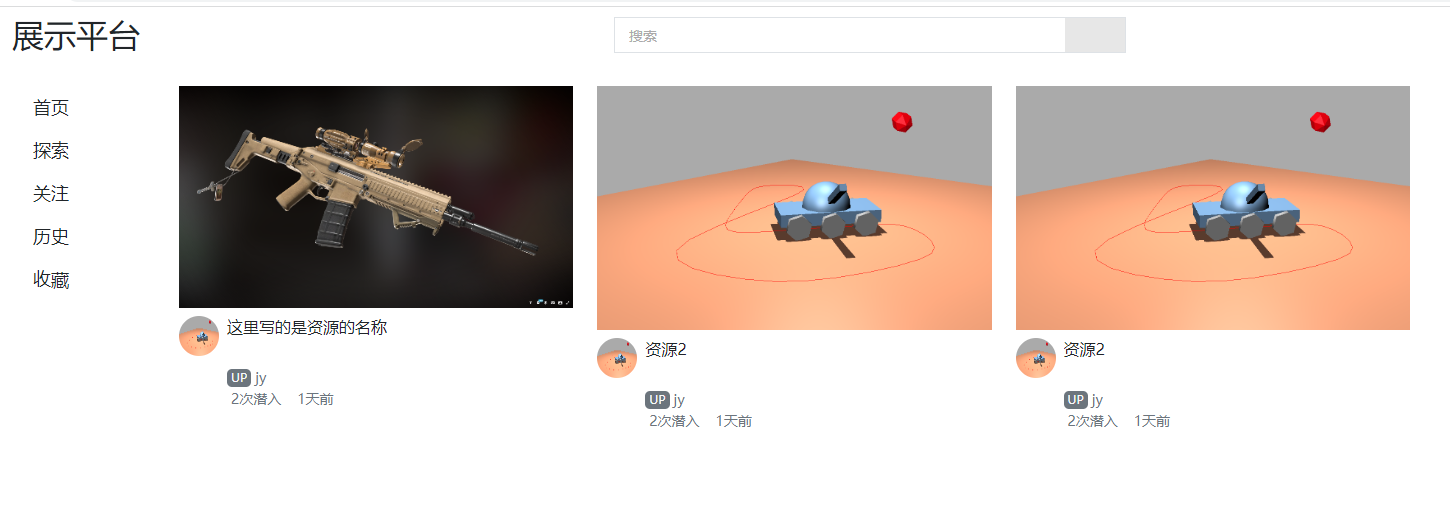
新增页面：



此页面只有上传按钮暂时有用，可以在主页场景新增一个场景页面（写死）

但是设计初衷是可以输入标题，以及上传图片作为封面，上传js文件，模型文件并进行新增。由于时间有限只进行到这，后续功能可以后续进行开发。

点击上传：



### 2.场景搭建

#### 场景1 设计可载入3d模型的页面，并可以进行镜头控制展示

在模型导入的场景情况下，可以在主展示区，用鼠标进行控制镜头的远近以及模型的拖动。

#### 场景2 使用three.js搭建一个场景

第二种场景是用threejs设计的场景，并本地化到vue中看是否可以运行。设计目的是可以适用于任意js场景。

## 三.实现

### 场景一

使用threejs的轨道控制器，并使用GLTFLoader来加载GLTF模型文件

将摄像机设置为60fov，0.01近面，1000远面来保证模型的正确显示

并设置初始位置

并把摄像机设置到轨道控制器中

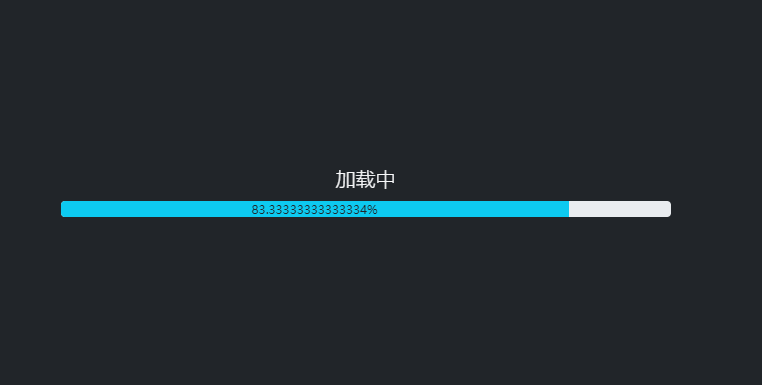
加载GLTFLoader进行模型加载

每帧都要监控整个轨道控制器的状态并进行渲染每次只移动5个单位

也就是画一个3d的三角形

坐标x+5，y+5 ,z-5

加载过程组件：



加载完成可以在页面中控制的情况：



监控模型的加载进度并实时反馈

代码：

<script>

import { onMounted, ref } from "vue";

import \* as THREE from "three";

import { OrbitControls } from "three/examples/jsm/controls/OrbitControls.js";

import { GLTFLoader } from "three/examples/jsm/loaders/GLTFLoader.js";

import {

  BloomEffect,

  EffectComposer,

  EffectPass,

  RenderPass,

} from "postprocessing";

export default {

  props: ["path"],

  setup(props) {

    const sceneRef = ref(null);

    const progress = ref(0);

    const flag = ref(false);

    console.log(props.path);

    let scene, renderer, camera, controls, spotLight, composer;

    const init = () => {

      // Scene

      scene = new THREE.Scene();

      scene.background = new THREE.Color(0x212529);

      // Renderer

      renderer = new THREE.WebGLRenderer({ antialias: true });

      renderer.setSize(sceneRef.value.clientWidth, sceneRef.value.clientHeight);

      renderer.toneMapping = THREE.ReinhardToneMapping;

      renderer.toneMappingExposure = 3;

      renderer.shadowMap.enabled = true;

      sceneRef.value.appendChild(renderer.domElement);

      // Camera

      const aspect = sceneRef.value.clientWidth / sceneRef.value.clientHeight;

      camera = new THREE.PerspectiveCamera(60, aspect, 0.01, 1000);

      camera.position.set(0, 3, 5);

      // Camera Controls

      controls = new OrbitControls(camera, renderer.domElement);

      controls.update();

      // Light

      const hemiLight = new THREE.HemisphereLight(0xffeeb1, 0x080820, 2);

      scene.add(hemiLight);

      spotLight = new THREE.SpotLight(0xffa95c, 4);

      spotLight.castShadow = true;

      spotLight.shadow.bias = -0.0001;

      spotLight.shadow.mapSize.width = 10000;

      spotLight.shadow.mapSize.height = 10000;

      scene.add(spotLight);

      // Post Processing

      composer = new EffectComposer(renderer);

      composer.addPass(new RenderPass(scene, camera));

      const effectPass = new EffectPass(

        camera,

        new BloomEffect({

          intensity: 3,

          luminanceThreshold: 0.8,

          width: 100,

          height: 100,

        })

      );

      composer.addPass(effectPass);

      // Loading Manager

      const manager = new THREE.LoadingManager();

      manager.onProgress = (url, itemsLoaded, itemsTotal) => {

        progress.value = (itemsLoaded / itemsTotal) \* 100;

        if ((itemsLoaded / itemsTotal) \* 100 === 100) {

          setTimeout(() => {

            flag.value = true;

          }, 1000);

        }

      };

      // Loader

      const loader = new GLTFLoader(manager);

      loader.load(props.path, (result) => {

        const model = result.scene.children[0];

        model.scale.set(1, 1, 1);

        model.traverse((n) => {

          if (n.isMesh) {

            n.castShadow = true;

            n.receiveShadow = true;

            if (n.material.map) n.material.map.anisotropy = 100;

          }

        });

        scene.add(model);

        animate();

      });

    };

    const animate = () => {

      requestAnimationFrame(animate);

      controls.update();

      spotLight.position.set(

        camera.position.x + 5,

        camera.position.y + 5,

        camera.position.z - 5

      );

      // renderer.render(scene, camera);

      composer.render();

    };

    const onCanvasResize = () => {

      camera.aspect = sceneRef.value.clientWidth / sceneRef.value.clientHeight;

      camera.updateProjectionMatrix();

      renderer.setSize(sceneRef.value.clientWidth, sceneRef.value.clientHeight);

      composer.setSize(sceneRef.value.clientWidth, sceneRef.value.clientHeight);

    };

    onMounted(() => {

      window.addEventListener("resize", onCanvasResize, false);

      init();

    });

    return {

      sceneRef,

      progress,

      flag,

    };

  },

};

</script>

### 场景二

本部分代码较多请移步源码

1.初始摄像机

摄像机初始看向世界原点(0,0,0)

2.光源阴影背景色的设置

3.活动平地

普通plan材质50\*50

设置rotation.x数学公式：π\*-.5

4.初始化坦克主体

轮子和炮台均为坦克主体的子scene

计算好轮子的相对位置、

[-carWidth / 2 - wheelThickness / 2, -carHeight / 2,  carLength / 3],

        [ carWidth / 2 + wheelThickness / 2, -carHeight / 2,  carLength / 3],

        [-carWidth / 2 - wheelThickness / 2, -carHeight / 2, 0],

        [ carWidth / 2 + wheelThickness / 2, -carHeight / 2, 0],

        [-carWidth / 2 - wheelThickness / 2, -carHeight / 2, -carLength / 3],

        [ carWidth / 2 + wheelThickness / 2, -carHeight / 2, -carLength / 3],

设置炮管跟随小球的世界坐标进行移动

设置炮管以炮台（球体）的中心为起始坐标进行旋转

炮管为坦克主体的子scene

5.设置小球

随着时间来在可定的区间范围内进行移动

6.跟踪每一帧的动态

time为0.001；

          time \*= 0.001;

          if (resizeRendererToDisplaySize(renderer)) {

              const canvas = renderer.domElement;

              cameras.forEach((cameraInfo) => {

                  const camera = cameraInfo.cam;

                  camera.aspect = canvas.clientWidth / canvas.clientHeight;

                  camera.updateProjectionMatrix();

              });

          }

          // move target移动目标物

          targetOrbit.rotation.y = time \* .27;

          targetBob.position.y = Math.sin(time \* 2) \* 4;

          targetMesh.rotation.x = time \* 7;

          targetMesh.rotation.y = time \* 13;

          targetMaterial.emissive.setHSL(time \* 10 % 1, 1, .25);

          targetMaterial.color.setHSL(time \* 10 % 1, 1, .25);

          // 移动坦克

          const tankTime = time \* .05;

          curve.getPointAt(tankTime % 1, tankPosition);

          curve.getPointAt((tankTime + 0.01) % 1, tankTarget);

          tank.position.set(tankPosition.x, 0, tankPosition.y);

          tank.lookAt(tankTarget.x, 0, tankTarget.y);

          // 炮管朝向

          targetMesh.getWorldPosition(targetPosition);

          turretPivot.lookAt(targetPosition);

          // 炮管摄像机朝向

          turretCamera.lookAt(targetPosition);

          // 目标物摄像机朝向

          tank.getWorldPosition(targetPosition);

          targetCameraPivot.lookAt(targetPosition);

          // 旋转

          wheelMeshes.forEach((obj) => {

              obj.rotation.x = time \* 3;

          });

          const camera = cameras[time \* .25 % cameras.length | 0];

          infoElem.textContent = camera.desc;

          renderer.render(scene, camera.cam);

          requestAnimationFrame(render);