**实 验 报 告 四**

**（2021-2022学年第一学期）**

**计算机图形学**

**（Computer Graphics）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 小组成员姓名 | 班级 | 学号 | | 成绩 |
| 1 |  |  |  | |  |
| 2 |  |  |  | |  |
| 3 |  |  |  | |  |
| 4 |  |  |  | |  |
| 5 |  |  |  | |  |
| 小组成绩： | | 任课教师签名： | | 日期： | |

|  |
| --- |
| **实验题目：** |
| **目录：**   1. 实验内容描述，即场景名称和特点的描述 2. 实验功能算法描述，即多个模型是如何建立、如何载入、如何观察的 3. 实验shader程序描述，即vertex shader和fragment shader的程序代码及说明 4. 其他功能描述，如交互、光照、纹理、类定义、基础库功能等 5. 实验结果，最终的场景实验结果图 6. 小组成员任务分工 |
| **实验报告：**   1. 实验内容描述，即场景名称和特点的描述   本次实验通过组合几个模型组成了场景。  我的场景由地板、位于中间的人物模型，和坐圆周运动的茶壶构成  地板为灰色大矩形，茶壶可以通过方向键旋转，点光源在场景上空做圆周运动  详见 (5)最终的场景实验结果图   1. 实验功能算法描述，即多个模型是如何建立、如何载入、如何观察的    1. 模型建立   茶壶模型为ThreeJS内置模型数据  人物模型为互联网上下载的GLTF模型   * 1. 模型载入   茶壶模型通过内置的顶点数据，构建为Object3D对象  人物模型通过GLTFLoader，载入为 Object3D数组   * 1. 模型观察   模型通过ThreeJS的 scene统一存储，通过renderer 控制渲染流程，通过camera渲染。   1. 实验shader程序描述，即vertex shader和fragment shader的程序代码及说明    1. vertex shader   varying vec3 Normal;  varying vec3 Position;  varying vec2 vUv;  void main() {  Normal = normalize(normalMatrix \* normal);  Position = vec3(modelViewMatrix \* vec4(position, 1.0));  vUv = uv;  gl\_Position = projectionMatrix \* modelViewMatrix \* vec4(position, 1.0);  }  Vs储存纹理贴图坐标，法向量，位置   * 1. fragment shader   varying vec3 Normal;  varying vec3 Position;  varying vec2 vUv;  uniform vec3 Ka;  uniform vec3 Kd;  uniform vec3 Ks;  uniform vec4 LightPosition;  uniform vec3 LightIntensity;  uniform float Shininess;  uniform sampler2D tt;  vec3 phong() {  vec3 n = normalize(Normal);  vec3 s = normalize(vec3(LightPosition) - Position);  vec3 v = normalize(vec3(-Position));  vec3 r = reflect(-s, n);  vec3 ambient = Ka;  vec3 diffuse = Kd \* max(dot(s, n), 0.0);  vec3 specular = Ks \* pow(max(dot(r, v), 0.0), Shininess);  return LightIntensity \* (ambient + diffuse + specular);  }  void main() {  gl\_FragColor = texture2D(tt, vUv) + vec4(phong(), 1.0);  }  Fs通过vs传递的数据计算光照强度与贴图像素颜色   1. 其他功能描述，如交互、光照、纹理、类定义、基础库功能等    1. 交互   Teapot的初始化函数中有如下定义  document.onkeydown = (e) => {  switch (e.keyCode) {  case 37:  this.rotation.x += 0.1  break  case 38:  this.rotation.z -= 0.1  break  case 39:  this.rotation.x -= 0.1  break  case 40:  this.rotation.z += 0.1  break  }  }  通过监听keydown事件，做到通过按键控制茶壶旋转。   * 1. 光照   每次渲染都会调用光照的update函数，以下是函数代码  update() {  this.theta = (this.theta + 0.01) % (Math.PI \* 2)  this.uniforms.LightPosition.value = new Vector4(  DIS \* Math.cos(this.theta),  DIS \* Math.sin(this.theta),  0.0,  1.0  )  }  this.uniforms.LightPosition 存储了shader中所需的光照数据   * 1. 纹理   因为ThreeJS中贴图与ShaderMaterial存在强绑定，且GLTF的模型中的贴图不方便提取，故所有模型统一使用一个贴图。以下为纹理贴图的局部：     * 1. 类定义   以下为核心部分的TypeScript代码（篇幅所限，已省去部分细节）：  export class LightMaterial extends ShaderMaterial {  theta: number = 0  constructor() {  super({  uniforms: {}, // shader 所需数据,  vertexShader,  fragmentShader,  })  }  update() {} // 计算旋转后的位置并更新  }  class Teapot extends Mesh {  theta: number  constructor() {  const geometry = new TeapotGeometry(30, 15)  super(geometry, new LightMaterial())  this.position.set(100, 100, 0)  // 注册键盘事件  }  update() {  // 计算圆周运动的位置并更新  }  }  class Floor extends Mesh {  constructor() {  const geometry = new PlaneBufferGeometry(2000, 2000, 8, 8)  const material = new MeshBasicMaterial({  color: 'grey',  side: DoubleSide,  })  super(geometry, material)  this.rotateX(-Math.PI / 2)  this.position.set(0, 0, 0)  }  }  export class App {  private readonly scene = new Scene()  private readonly renderer = new WebGLRenderer({  antialias: true,  precision: 'highp',  canvas: document.getElementById('main-canvas') as HTMLCanvasElement,  })  private readonly camera = new PerspectiveCamera(  45,  innerWidth / innerHeight,  1,  10000  )  private readonly controls = new OrbitControls(  this.camera,  this.renderer.domElement  )  private readonly teapot = new Teapot()  private readonly floor = new Floor()  private gltf?: GLTF = undefined  constructor() {  this.scene.add(this.teapot)  this.scene.add(this.floor)  const loader = new GLTFLoader()  loader.load('./static/exp-1-gltf/scene.gltf', gltf => {  this.gltf = gltf  gltf.scene.traverse(obj => {  if (obj instanceof Mesh) obj.material = new LightMaterial()  })  this.scene.add(gltf.scene)  })  this.controls.target.set(0, 0, 0)  this.controls.object.position.set(300, 300, 300)  this.controls.update()  this.renderer.setSize(innerWidth, innerHeight)  this.renderer.setClearColor(new Color('rgb(223, 199, 254)'))  this.render()  }  private adjustCanvasSize() {  this.renderer.setSize(innerWidth, innerHeight)  this.camera.aspect = innerWidth / innerHeight  this.camera.updateProjectionMatrix()  }  private render() {  this.teapot.update()  ;(this.teapot.material as LightMaterial).update()  this.gltf?.scene.traverse(obj => {  if (obj instanceof Mesh) (obj.material as LightMaterial).update()  })  this.renderer.render(this.scene, this.camera)  this.adjustCanvasSize()  requestAnimationFrame(() => this.render())  }  }   1. 实验结果，最终的场景实验结果图         可以观测到茶壶位置、朝向有变化，人物模型的光照方向发生了改变。   1. 小组成员任务分工   个人小组，全部工作由本人完成 |