电 子 科 技 大 学

实　验　报　告

学生姓名：向睿　　学 号：2023090911028　　指导教师：何明耘

实验地点：信软楼　　实验时间：2024.11.30

一、实验室名称： 软件信息技术专业实验室

二、实验项目名称：人体动画展现

三、实验学时： 4

四、实验原理：

数字人技术（Digital Human Technology）是计算机图形学和人工智能领域的交叉学科，它涉及到虚拟人物的创建、动画设计、交互控制以及与用户的实时互动。数字人技术的核心目标是通过虚拟人物的逼真表现和行为，使得虚拟人物能够与真实世界进行互动，广泛应用于娱乐、游戏、教育、医疗等领域。

1. 数字人的构建与动画原理

数字人的构建通常包括模型的创建和动画的设计。在本实验中，数字人（Avatar）是通过三维建模技术实现的，使用 3D 建模建模网站 Ready Player Me 来创建数字人的外形、骨骼结构、衣物和面部表情等内容。然后，借助 3D 引擎（three.js）将其转化为可交互、可动画的数字人物。

数字人模型的骨骼系是动画的基础，骨骼动画技术允许通过控制模型的骨骼结构，来驱动整个模型进行动画变换。常见的骨骼动画类型包括行走、跑步、跳跃等动作。骨骼动画的核心原理是将复杂的模型动作分解为多个骨骼的运动，每个骨骼的变换通过一定的算法进行插值，最终实现自然流畅的动作过渡。

2. 三维引擎（Three.js）的应用原理

three.js 是一个基于 WebGL 的 JavaScript 库，旨在简化 Web 上的三维图形开发。它能够

通过编程语言动态生成 3D 场景、模型、光源、相机、动画等，并将其渲染到浏览器中，支持丰富的互动功能。Three.js 使得开发者可以在浏览器中方便地实现三维图形、虚拟现实以及交互式应用。

在本实验中，three.js 用于加载并显示数字人模型，动画控制，视点管理等任务。通过 three.js 的 GLTFLoader 和 AnimationMixer 等功能，我们可以加载 GLTF 格式的数字人模型，并为其添加动画效果。GLTF 是近年来广泛使用的开放标准格式，能够较为高效地表示 3D 模型及其动画信息。

3. 数字人交互控制原理

数字人的交互性是指数字人物能够响应用户的输入并产生相应的行为。在本实验中，交互控制通过键盘和鼠标实现。通过键盘事件监听，用户可以控制数字人的动作，比如让其进行行走、跑步、跳跃等。通过鼠标事件控制，用户可以操作视点，从而调整摄像机的位置，改变数字人模型的观察角度。

交互控制中涉及的关键技术是事件监听与状态管理。当用户按下特定的键或点击鼠标时，程序会触发相应的事件，通过设置动画的权重或切换动画来实现数字人对不同输入的响应。同时，通过平滑过渡技术，使得数字人从一个动作切换到另一个动作时能够保持流畅。

4. 多个动画的融合

在数字人模型的动画控制中，多个动作的融合是非常重要的一部分。动画融合是指将多个动画效果以不同的权重混合，从而在不同的情境下实现自然的过渡。例如，数字人在跑步过程中如果突然开始行走，则需要平滑过渡，而非直接跳跃到行走状态。

Three.js 中的 AnimationMixer 类提供了强大的动画融合功能。通过设置不同动画的权重，我们可以控制数字人模型不同部分的动作强度，实现多种动作的平滑过渡。此外，权重的变化可以根据外部输入（如键盘事件）实时调整，进而达到更为自然的动画切换效果。

5. Morph Target 动画

Morph Target 动画（形态目标动画）是一种通过修改模型顶点位置来实现动画的技术。与骨骼动画不同，Morph Target 动画的原理是通过控制模型的顶点位置，创建多个目标形态，然后通过插值计算，生成模型在不同形态之间的过渡。Morph Target 动画常用于表现细节变化，如面部表情、肌肉运动等。

Morph Target 技术可以用于数字人的面部表情变化或细微动作的表现。通过在模型中定义多个形态目标（例如微笑、皱眉等），然后通过设置权重，控制不同形态目标的混合比例，从而实现更加细腻的动画效果

五、实验目的：

1） 数字人技术的了解，通过实验进行实践认知，为后面的实验做一个知识准备

2） 相关引擎的应用，包括建模，动画设计，threejs 编程

3） 数字人交互技术的实现，利用鼠标/摄像头进行交互

4） 了解相关数字人产业发展以及相关底层 AI 技术的发展

六、实验内容：

1） 创建一个 avatar，了解创建过程

2） 为 avatar 设置相关动画，至少创建 2 个以上动画

3） 利用 threejs 编程数字人基本展示

4） 利用 threejs 实现数字人交互（可选择其他 https://ift.devinci.fr/3D-interactive-avatar ）

a) 键盘进行动作选择

b) 鼠标进行视点控制

5） 多个动作融合

6） 了解 morph target 动画

七、实验器材（设备、元器件）：

1） 多媒体计算机

2） Vs Code

3） three.js 开发平台。

4） 3D 模型（avatar 模型、动作文件等）。

5） GLTFLoader 库，鼠标和键盘输入设备。

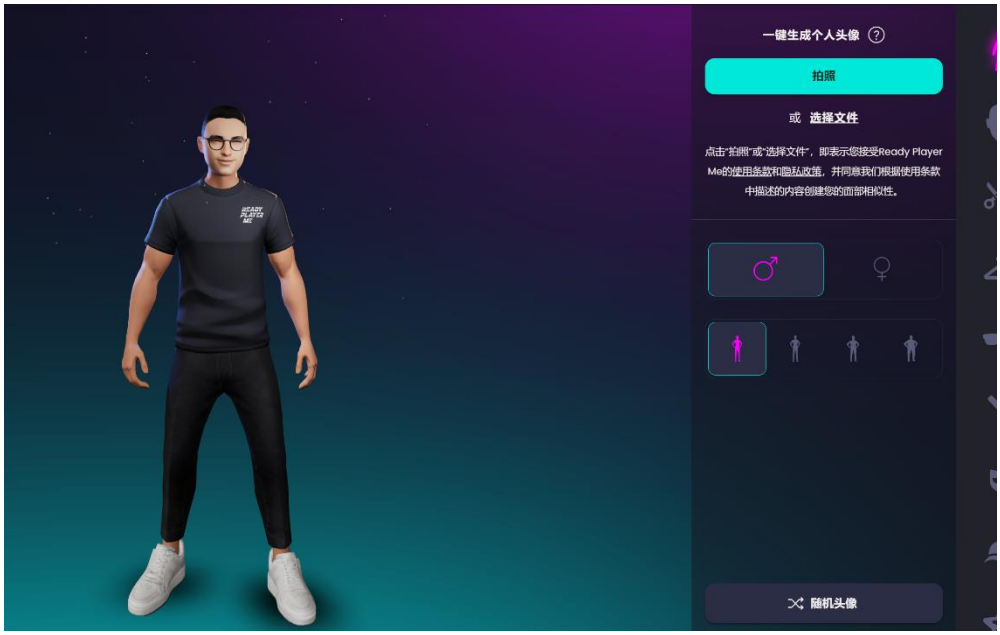
八、实验步骤：

步骤 1 avatar 创建

1） 生成模型

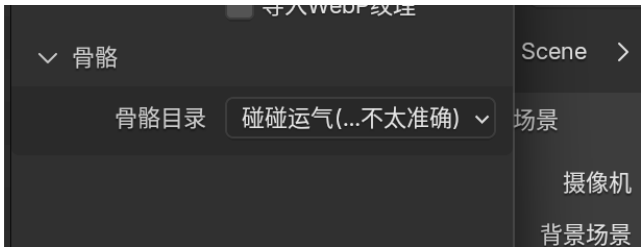
在 Ready Player Me 网站上通过自拍或上传照片生成自己的模型，并选择

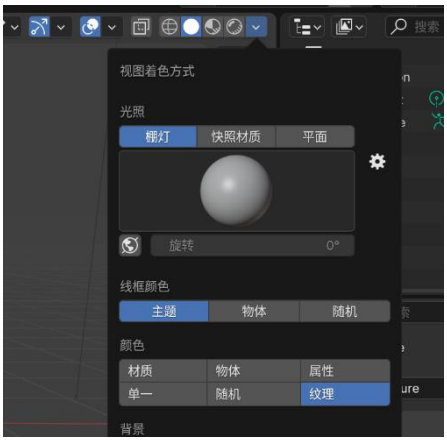
喜欢的搭配，点击 next 下载 glb 格式文件。

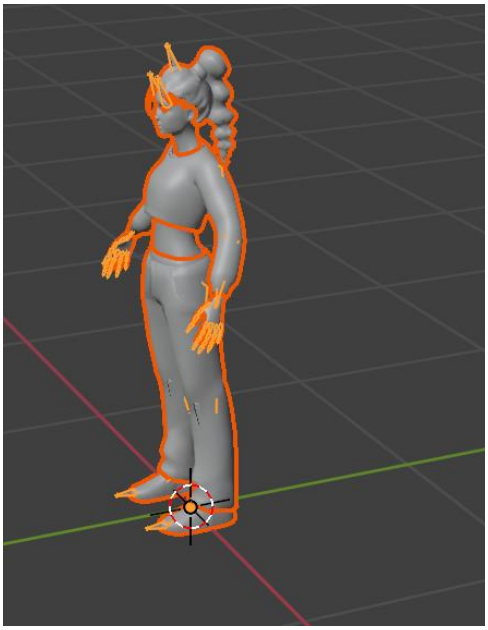


2） glb 转换为 fbx 文件

a) 将 glb 文件导入 blender，骨骼选择碰碰运气。导入后颜色选择纹理，即出现服装。



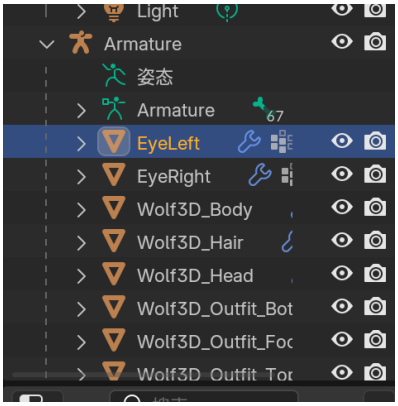




【blender 相关操作】

1. 按住滚轮移动鼠标——调整视角。
2. 滑动滚轮——缩放
3. 移动小手——调整视图

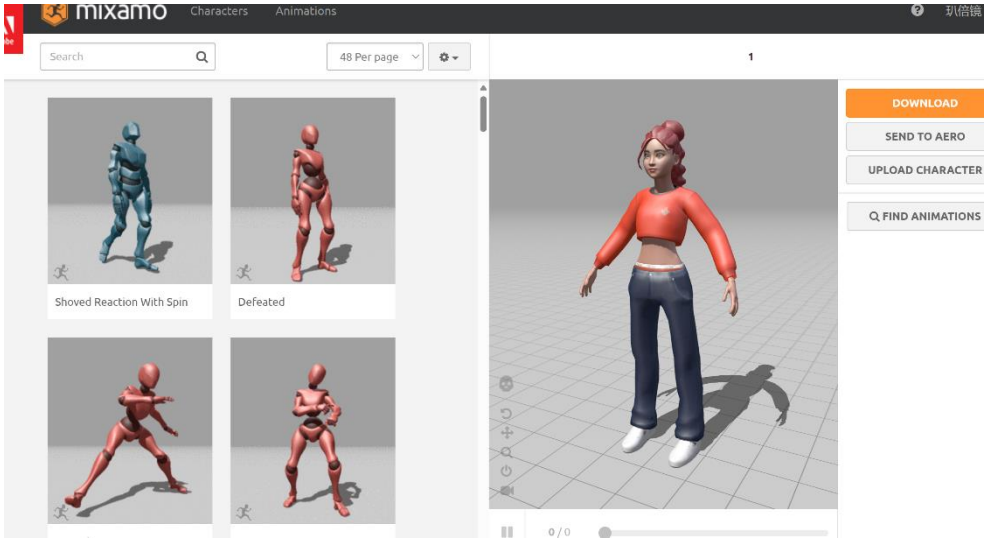
b) 将纹理内嵌。工具栏选择“着色”，选中 EyeLeft，找到下方的 BASE COLOR，点击解包项，选择将文件写至原目录（覆盖现有文件），Amature 下面其他项重复此操作。



c) 导出 fbx 文件。选择文件，导出，fbx。路径模式选择复制，打开内嵌纹理。导出。

步骤 2　为 avatar 设置相关动画

1） 将 fbx 文件导入 mixamo



2） 在左边选择合适动画，然后下载

步骤 3　利用 threejs 实现数字人基本展示

1） 初始化

```
function init() {
  //1.获取容器
  const container = document.getElementById('container')


  //2.场景
  scene = new THREE.Scene()
  scene.background = new THREE.Color(0xebd2c3)
  //添加雾：将地板颜色和背景颜色模糊在一起
  //scene.fog = new THREE.Fog(0xffffff, 10, 50)

  //3.相机
  camera = new THREE.PerspectiveCamera(50, window.innerWidth / window.innerHeight, 0.1, 10
  camera.position.z = 30
  camera.position.x = 0;
  camera.position.y = -3;
  //4.渲染器
  renderer = new THREE.WebGLRenderer({ antialias: true })//antialias是否抗锯齿
  renderer.setPixelRatio(window.devicePixelRatio)//设置像素比。避免HIDPI上面 绘图模糊
  renderer.setSize(window.innerWidth, window.innerHeight)//将输出的canvas调整为网页大小
  renderer.shadowMap.enabled = true//投射阴影
  document.body.appendChild(renderer.domElement)//在body上添加canvas并将其渲染
```

2） 添加光照

a) 半球光源

b) 平行光

```
//5.光照
//5.1环境光
/* 半球光源
    上半球白色，下半球灰色 */
const hemiLight = new THREE.HemisphereLight(0xffffff, 0xffffff, 1.7)
hemiLight.position.set(0, 50, 0)
scene.add(hemiLight)
//5.2平行光
/* 模拟太阳光等远距离的强光源，光线是平行的。
光线白色，强度3 */
const dirLight = new THREE.DirectionalLight(0xffffff, 1)
dirLight.position.set(-8, 12, 12)
//5.3设置阴影相机的裁剪面
/* 平行光使用阴影相机来控制哪些区域会被阴影覆盖。
通过 dirLight.shadow.camera 属性设置阴影相机的裁剪面 */
/* top、bottom、left、right：这些参数控制阴影相机的视野范围，确定光源能够照射到的区域大小。
near 和 far：这两个参数控制阴影相机的近裁剪面和远裁剪面，决定了阴影的可见范围。 */
let d = 8.25;
dirLight.castShadow = true;
dirLight.shadow.mapSize = new THREE.Vector2(1024, 1024);
dirLight.shadow.camera.near = 0.1;
dirLight.shadow.camera.far = 1500;
dirLight.shadow.camera.left = d * -1;
dirLight.shadow.camera.right = d;
dirLight.shadow.camera.top = d;
dirLight.shadow.camera.bottom = d * -1;
scene.add(dirLight)
```

3） 地面

```
//6.地面
let floorGeometry = new THREE.PlaneGeometry(5000, 5000, 1, 1);//足够大平面
let floorMaterial = new THREE.MeshPhongMaterial({
  color: 0xebd2c3,
  shininess: 0,
});
let floor = new THREE.Mesh(floorGeometry, floorMaterial);
floor.rotation.x = -0.5 * Math.PI;
floor.receiveShadow = true;
floor.position.y = -11;
scene.add(floor);
```

4） 轨道控制器

```
//9.轨道控制器
const controls = new OrbitControls(camera, renderer.domElement);
controls.enablePan = false;
controls.enableZoom = false;
controls.target.set(0, 1, 0);
controls.update();
```

5） 导入模型

```
/* TODO3:加载模型 */
//7.加载模型
//7.1创建实例
const loader = new GLTFLoader()

//7.2加载模型
loader.load('./images/animation.glb', gltf => {
  //* 检查是否导入成功 */
  //console.log(gltf.scene)
  //console.log(gltf.animations)

  scene.add(gltf.scene);
  //7.3添加模型

  model = gltf.scene
  const animations = gltf.animations;// 提取加载的模型中包含的动画数据
  //7.4遍历所有子物体

  //为每个网格设置阴影
  model.traverse(child => {
    if (child.isMesh) {
      child.castShadow = true;
      child.receiveShadow = true;
    }
    if (child.isBone && child.name === 'Neck') {
      neck = child; // 颈部
    }
    if (child.isBone && child.name === 'Spine') {
      waist = child; // 腰部
    }
  })
  //将模型比例设置为初始的7倍
  model.scale.set(7, 7, 7)
  //移动模型
  model.position.y = -11
  scene.add(model)
```

步骤 4　实现数字人交互

1） 鼠标进行视点控制

```
/* TODO4.2:鼠标进行视点控制 */
//监听鼠标事件
document.addEventListener('mousemove', function (e) {
  const mousecoords = getMousePos(e)
  if (neck && waist) {
    moveJoint(mousecoords, neck, 50)
    moveJoint(mousecoords, waist, 30)
  }
})
//获取鼠标坐标
function getMousePos(e) {
  return { x: e.clientX, y: e.clientY }
}
//移动关节
function moveJoint(mouse, joint, degreeLimit) {
  let degrees = getMouseDegrees(mouse.x, mouse.y, degreeLimit)
  joint.rotation.y = THREE.MathUtils.degToRad(degrees.x)
  joint.rotation.x = THREE.MathUtils.degToRad(degrees.y)

}
//调整对应旋转角度
```

```
//调整对应旋转角度
function getMouseDegrees(x, y, degreeLimit) {
  let dx = 0,
    dy = 0,
    xdiff,
    xPercentage,
    ydiff,
    yPercentage

  let w = { x: window.innerWidth, y: window.innerHeight
  if (x <= w.x / 2) {
    xdiff = w.x / 2 - x
    xPercentage = (xdiff / (w.x / 2)) * 100
    dx = ((degreeLimit * xPercentage) / 100) * -1
  }
  if (x >= w.x / 2) {
    xdiff = x - w.x / 2
    xPercentage = (xdiff / (w.x / 2)) * 100
    dx = (degreeLimit * xPercentage) / 100
  }
  if (y <= w.y / 2) {
    ydiff = w.y / 2 - y

    yPercentage = (ydiff / (w.y / 2)) * 100
    dy = (((degreeLimit * 0.5) * yPercentage) / 100) *
  }
  if (y >= w.y / 2) {
    ydiff = y - w.y / 2
    yPercentage = (ydiff / (w.y / 2)) * 100
    dy = (degreeLimit * yPercentage) / 100

  }
  return {
    x: dx, y: dy
  }
}
```

```
//更新函数
function update() {
  renderer.render(scene, camera)
  requestAnimationFrame(update)

  if (resizeRendererToDisplaySize(renderer)) {
    const canvas = renderer.domElement;
    camera.aspect = canvas.clientWidth / canvas.clientHeight;
    camera.updateProjectionMatrix();
  }
}
update()
//保持模型比例
function resizeRendererToDisplaySize(renderer) {
  const canvas = renderer.domElement;
  let width = window.innerWidth;
  let height = window.innerHeight;
  let canvasPixelWidth = canvas.width / window.devicePixelRatio;
  let canvasPixelHeight = canvas.height / window.devicePixelRatio;

  const needResize =
    canvasPixelWidth !== width || canvasPixelHeight !== height;
  if (needResize) {
    renderer.setSize(width, height, false);
  }
  return needResize;
}
```

2） 键盘进行控制

```
/* TODO4.1:键盘进行控制 */
//监听键盘事件
document.addEventListener('keydown', (e) => {
  let time = 0

  if (e.keyCode === 40 || e.keyCode === 39) {//按向下或向右键切换

    index >= actions.length - 1 ? index = 0 : index++;

    prepareCrossFade(currentAction, actions[index], time)//切换下一个动画
    currentAction = actions[index]
  }
  if (e.keyCode === 38 || e.keyCode === 37) {//向左或向上键切换
    index === 0 ? index = actions.length - 1 : index--;
    prepareCrossFade(currentAction, actions[index], time)//切换前一个动画
    currentAction = actions[index]
  }
  //console.log(currentAction)
})
```

步骤 5　多个动作融合

1） 获取动画并分别命名

我有四个动作，分别是 breakdance 中的 freeze，footwork，ending 以及一个站立动画

```
mixer = new THREE.AnimationMixer(model)
standAction = mixer.clipAction(animations[6])
freezeAction = mixer.clipAction(animations[5])
footworkAction = mixer.clipAction(animations[4])
endingAction = mixer.clipAction(animations[0])



actions = [standAction, freezeAction, footworkAction, endingAction]
console.log(animations)
activateAllActions()


renderer.setAnimationLoop(animate)
```

2） 创建控制面板

```
//创建用户控制面板
function createPanel() {
  const panel = new GUI({ width: 310 });
  //添加folder
  const folder1 = panel.addFolder('Visibility');//控制骨骼可见性
  const folder2 = panel.addFolder('Activation/Deactivation');//控制动画启用于禁用
  const folder3 = panel.addFolder('Pausing/Stepping');//暂停/继续动画
  const folder4 = panel.addFolder('Crossfading');//控制动画渐变过渡
  const folder5 = panel.addFolder('Blend Weights');//调整不同动画权重
  const folder6 = panel.addFolder('General Speed');//调整动画播放速度
```

3） 设置控制项

```
//设置控制项
settings = {
    'show model': true,
    'show skeleton': false,
    'deactivate all': deactivateAllActions,
    'activate all': activateAllActions,
    'pause/continue': pauseContinue,
    'make single step': toSingleStepMode,//单步播放
    'modify step size': 0.05,
    'freeze': function () {
```

a) 播放函数

```
'freeze': function () {
    prepareCrossFade(currentAction, freezeAction, 0);
    currentAction = freezeAction;
},
'stand': function () {

    prepareCrossFade(currentAction, standAction, 0);
    currentAction = standAction
},
'footwork': function () {

    prepareCrossFade(currentAction, footworkAction, 0);
    currentAction = footworkAction
},

'ending': function () {
    prepareCrossFade(currentAction, endingAction, 0)
    currentAction = endingAction
},
```

b) 控制不同动画权重

```
//控制不同动画权重
folder5.add(settings, 'modify stand weight', 0.0, 1.0, 0.01).listen().onChange(fu

    setWeight(standAction, weight);

});
folder5.add(settings, 'modify freeze weight', 0.0, 1.0, 0.01).listen().onChange(f

    setWeight(freezeAction, weight);

});
folder5.add(settings, 'modify footwork weight', 0.0, 1.0, 0.01).listen().onChange

    setWeight(footworkAction, weight);

});

folder5.add(settings, 'modify ending weight', 0.0, 1.0, 0.01).listen().onChange(f

    setWeight(footworkAction, weight);

});
```

4） 关于动画过渡的一些函数

```
//从一个动画过渡到另一个动画
function prepareCrossFade(startAction, endAction, defaultDuration) {
    const duration = setCrossFadeDuration(defaultDuration);//设置淡出时间
    singleStepMode = false;
    unPauseAllActions();
    executeImmediateTransition(startAction, endAction, duration);
}
function executeImmediateTransition(startAction, endAction, duration) {
    // 直接停止当前动画
    deactivateAllActions()
    // 立即播放目标动画
    endAction.time = 0;
    setWeight(endAction, 1); // 设置权重为1，确保目标动画完全播放
    endAction.play(); // 直接播放目标动画
}
function setCrossFadeDuration(defaultDuration) {
    if (settings['use default duration']) {
        return defaultDuration;
    } else {
        return settings['set custom duration'];
    }
}
//同步两个动画动作切换
function synchronizeCrossFade(startAction, endAction, duration) {
    mixer.addEventListener('loop', onLoopFinished);
    function onLoopFinished(event) {
        if (event.action === startAction) {
            mixer.removeEventListener('loop', onLoopFinished);
            executeCrossFade(startAction, endAction, duration);
        } }
}
//淡入淡出动画
function executeCrossFade(startAction, endAction, duration) {
    setWeight(endAction, 1);
    endAction.time = 0;
    startAction.crossFadeTo(endAction, duration, true);
}
```

5） 关于权重改变的一些函数

```
//设置权重
function setWeight(action, weight) {
    action.enabled = true;
    action.setEffectiveTimeScale(1);
    action.setEffectiveWeight(weight);
}
//确保settings中权重值正确更新
function updateWeightSliders() {
    settings['modify stand weight'] = standAction;
    settings['modify freeze weight'] = freezeAction;
    settings['modify footwork weight'] = footworkAction;
    settings['modify ending weight'] = endingAction;
}
//启用或禁用某些按钮
function updateCrossFadeControls() {
    if (standWeight === 1 && freezeWeight === 0 && footworkWeight === 0 && endingWeig
        crossFadeControls[0].disable();
        crossFadeControls[1].enable();
        crossFadeControls[2].enable();
        crossFadeControls[3].enable();
    }
    if (standWeight === 0 && freezeWeight === 1 && footworkWeight === 0 && endingWeig
    }
    if (standWeight === 0 && freezeWeight === 0 && footworkWeight === 1 && endingWeig
    }

    if (standWeight === 0 && freezeWeight === 0 && footworkWeight === 0 && endingWeig
    }
}
//实现权重调节
function setWeightinit() {
    setWeight(actions['stand'], settings['modify stand weight'])
    setWeight(actions['freeze'], settings['modify freeze weight'])
    setWeight(actions['footwork'], settings['modify footwork weight'])
    setWeight(actions['ending'], settings['modify ending weight'])
}
```

步骤 6　结果展示

步骤 7　了解 morph target 动画

1） 基本概念

Morph Target 动画是一种通过改变物体的形状来实现动画的技术。它的核心思想是通过改变模型不同状态下的顶点位置来实现动画效果，而不是使用骨骼或其他方式。通常，Morph Target 动画用于面部表情、角色身体的形变、物体的软体变形等。

2） 工作原理

Morph Target 动画的工作原理是通过插值法在多个不同的形状之间过渡。这些形状被称为 目标形状，每个目标形状代表了一个物体的状态。

a) 目标形状：每个 Morph Target 存储了顶点位置的变更，它是基于初始模型的某种变化。例如，面部表情的不同（微笑、皱眉等），或是角色的不同姿势（站立、跑步等）。

b) 权重控制：每个目标形状都有一个 权重值，控制目标形状对最终显示结果的影响。通过调整这些权重值，你可以控制形状从一个状态过渡到另一个状态。

3） 与其他动画的区别

与传统的骨骼动画不同，Morph Target 动画并不依赖于骨骼或关节。它直接操作模型的顶点位置，适合于需要局部形变的场景。

例如，在 面部动画 中，Morph Target 可以通过控制嘴巴、眼睛等部分的形状来实现各种表情的变化，而无需改变模型的骨骼结构。

4） 应用场景

a) 面部表情：通常用于角色的面部动画，如微笑、皱眉、眼睛眨动等。

b) 物体变形：适用于例如橡胶、液体等软体物体的形变。

c) 特殊效果：如角色在不同状态下的表现（变胖、变瘦等）。

5） 优缺点

a) 优点：

精细控制：可以直接控制模型的顶点位置，非常适合需要细致形变的场景。

独立于骨骼：不需要复杂的骨骼结构，可以直接在顶点层面上进行动画。

b) 缺点：

计算成本较高：相比于骨骼动画，Morph Target 需要存储多个形状的顶点数据，可能会占用更多的内存。

动画过渡可能不自然：如果目标形状设计不合理，可能会导致动画效果不够自然，尤其是形状之间的过渡不平滑时。

6） 在 3D 软件中的使用

在 3D 建模软件(如 Blender、Maya 等)中，Morph Target 通常被称为 Shape Keys 或 Blend Shapes。在这些软件中，用户可以通过修改顶点位置来创建不同的形状，并且控制它们之间的过渡。

例如，在 Blender 中，可以通过以下步骤创建 Morph Target 动画：

创建一个基础模型（例如，一个人物的脸）。

为该模型创建多个不同的表情（如笑、哭、惊讶等）。

控制每个表情的影响力（通常是通过滑块调整）。

7） 与骨骼动画结合使用

在许多复杂的动画场景中，Morph Target 动画与骨骼动画可以结合使用。例如，角色的骨骼控制全身的姿势，而 Morph Target 控制面部表情，二者结合使用可以实现更加自然和丰富的动画效果。

九、实验数据及结果分析：

在本实验中，我们主要通过 three.js 实现了数字人的基本展示、动画设置和交互功能，最终得到了能够通过键盘和鼠标控制的数字人模型。实验过程中，我们使用了 GLTFLoader 加载模型，使用 AnimationMixer 控制动画，并通过事件监听实现了键盘和鼠标的交互控制。

1） **数据与实验结果：**

a) **模型加载与展示：**

成功通过 three.js 加载了 GLTF 格式的数字人模型，并在浏览器中正确渲染。通过摄像机控制，我们能够对模型进行不同角度的查看，确保模型的显示无误。

b) **动画设置与控制：**

为数字人模型添加了至少两种动画，我添加的是 breakdance 的几个动作。通过 AnimationMixer 实现了动画的切换和权重控制。在动画切换时，能

够平滑过渡，避免生硬的切换效果。各动画的融合效果良好，可以通过按键选择不同的动作。

c) **交互功能：**

键盘事件实现了数字人动作的控制，鼠标事件成功控制了视点的位置与角度，用户能够通过鼠标的拖动操作调整视角，查看模型的不同部分。

2） **问题与解决：**

a) **问题一：动画切换卡顿**

在第一次实现动画切换时，发现数字人模型在切换动画时出现卡顿现象。分析发现问题出在动画的权重切换上，初始设置的权重值不适当。解决方案是优化了 AnimationMixer 中的权重调整逻辑，确保动画过渡更加平滑。

b) **问题二：交互响应延迟**

初始的鼠标视角控制存在一定的延迟，原因可能是在处理鼠标事件时计算模型视角的过程过于繁重。通过优化鼠标事件处理和视点更新的频率，最终解决了这一问题。

c) **问题三：浏览器自动缓存**

浏览器存在自动缓存的情况，每次更改后需清空缓存再打开页面，非常麻烦。通过安装 live-server 包，使用 live-server 命令，使每次更改后浏览器自动刷新。

d) **问题四：模型加载成功却不显示**

因为未将纹理内嵌，导致控制模型加载成功却不显示。通过使用 blender 将纹理内嵌，成功显示模型。

3） **结果分析：**

通过本次实验，我们能够成功创建一个可交互的数字人模型，并为其添加了丰富的动画效果。结果表明，使用 three.js 和 GLTF 格式模型能够高效地展示 3D 人物，并且通过合理的动画融合与权重控制，使得多个动画之间的切换更加自然，数字人的交互性得到了很好的体现。

十、实验结论：

通过本次实验，达到了以下几个关键目标：

1. **数字人建模与动画的实现：**

   成功使用 three.js 框架加载了 3D 模型，并为模型添加了骨骼动画。通过合理的动画融合技术，使得不同动画效果能够自然过渡，增强了数字人的表现力。

2. **交互功能的实现：**

   通过键盘和鼠标事件监听，成功实现了数字人模型的交互功能。用户能够通过键盘切换不同动作，并通过鼠标调整视角。数字人能够灵活响应不同的输入，提升了互动体验。

3. **数字人技术的前景：**

   数字人技术作为虚拟现实和增强现实中的重要应用，正在快速发展。本实验为数字人技术的学习和应用提供了基础知识，并通过实践了解了数字人建模、动画、交互控制等方面的核心技术。

十一、总结及心得体会：

通过本次实验，我对数字人技术有了更深入的理解，并通过实际编程实践掌握了基本的数字人模型加载、动画设置和交互控制。整个实验过程中，遇到了一些挑战和问题，但通过调试和优化，我成功解决了这些问题，并获得了最终的实验结果。

1. 数字人技术的挑战：

在进行数字人建模和动画设计时，最具挑战性的是如何使不同动画之间过渡自然。尤其是多个动画的融合和权重控制，如何平衡动画的自然度和用户输入的即时响应，是实验中的一个关键点。通过三维引擎中的 AnimationMixer 和适当的动画权重调整，最终达到了令人满意的效果。

2. 交互性的重要性：

交互性是数字人技术中的一个重要环节，实验中通过键盘和鼠标事件的结合，使得数字人能够根据用户的输入做出相应的动作变化。交互的流畅性对用户体验至关重要，因此需要在编程过程中充分优化事件响应的速度与精度。

3. 实验的收获与展望：

本次实验让我更加熟悉了 three.js 以及数字人建模与动画的基本原理。通过实践，我深入理解了 3D 模型加载、动画控制和交互设计的工作流程。未来，我

希望能够将这些知识应用到更复杂的项目中，例如为数字人加入语音识别、情感分析等功能，使其更加智能化，能够与用户进行更加丰富的互动。

总的来说，本次实验不仅加深了我对数字人技术的理解，也培养了我在三维图形编程和交互设计方面的实际能力。它为我未来在虚拟现实、游戏开发以及人工智能领域的学习和研究打下了良好的基础。

十二、对本实验过程及方法、手段的改进建议：

1. **实验流程优化：**

   在实验过程中，部分环节的调试较为繁琐，如动画的切换与融合部分。可以提前对三维引擎的基本操作和动画处理有更深入的了解，减少实验中的调试时间。

2. **多样化的交互设计：**

   本实验主要依赖键盘和鼠标作为输入设备，可以考虑扩展到其他交互方式，例如利用摄像头进行面部表情识别、手势控制等，这样能够进一步提升数字人交互的智能化和多样性。

3. **AI 集成与智能交互：**

   将来可以将 AI 技术集成到数字人的交互中，增强其智能化水平。比如通过语音识别技术使数字人能理解并执行用户的语音指令，或通过情感分析调整数字人的表情和行为，使其能够模拟更丰富的人类交互方式。

报告评分：　　XX

指导教师签字：XXXX