<!DOCTYPE html>

<html lang="en">

  <head>

    <title>three.js webgl - animation - skinning - ik</title>

    <meta charset="utf-8" />

    <meta

      name="viewport"

      content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0"

    />

    <meta name="author" content="Antoine BERNIER (abernier)" />

    <link type="text/css" rel="stylesheet" href="main.css" />

    <style>

      body {

*color*: white;

      }

      #info a {

*color*: #4d6675;

      }

    </style>

  </head>

  <body>

    <div id="info">

      <a href="https://threejs.org" target="\_blank" rel="noopener">three.js</a>

      - webgl - inverse kinematics<br />

      Character model by

      <a

        href="https://assetstore.unity.com/packages/3d/characters/humanoids/humans/kira-lowpoly-character-100303"

        target="\_blank"

        rel="noopener"

        >Aki</a

      >, furnitures from

      <a href="https://poly.pizza" target="\_blank" rel="noopener">poly.pizza</a

      >, scene by

      <a

        href="https://abernier.name/three.js/examples/webgl\_esher.html"

        target="\_blank"

        rel="noopener"

        >abernier</a

      >. CC0.

    </div>

    <script type="importmap">

      {

        "imports": {

          "three": "./js/three.js-master/build/three.module.min.js",

          "three/addons/": "./js/three.js-master/examples/jsm/"

        }

      }

    </script>

    <script type="module">

      import \* as THREE from "three";

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

      import { TransformControls } from "three/addons/controls/TransformControls.js";

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

      import { DRACOLoader } from "three/addons/loaders/DRACOLoader.js";

      import {

        CCDIKSolver,

        CCDIKHelper,

      } from "three/addons/animation/CCDIKSolver.js";

      import Stats from "three/addons/libs/stats.module.js";

      import { GUI } from "three/addons/libs/lil-gui.module.min.js";

*let* scene, camera, renderer, orbitControls, transformControls;

*let* mirrorSphereCamera;

*const* OOI = {};

*let* IKSolver;

*let* stats, gui, conf;

*const* v0 = new THREE.Vector3();

      init();

      async *function* init() {

        conf = {

          followSphere: false,

          turnHead: true,

          ik\_solver: true,

          update: updateIK,

        };

        scene = new THREE.Scene();

        scene.fog = new THREE.FogExp2(0xffffff, 0.17);

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

        camera = new THREE.PerspectiveCamera(

          55,

          window.innerWidth / window.innerHeight,

          0.001,

          5000

        );

        camera.position.set(1, 1.5, 2); // Ajusta la posición de la cámara para una mejor vista

        camera.lookAt(scene.position);

*const* ambientLight = new THREE.AmbientLight(0xffffff, 8); // Luz blanca suave

        scene.add(ambientLight);

*const* directionalLight = new THREE.DirectionalLight(0xffffff, 5);

        directionalLight.position.set(5, 10, 7.5); // Ajusta la dirección de la luz

        scene.add(directionalLight);

*const* dracoLoader = new DRACOLoader();

        dracoLoader.setDecoderPath(

          "./js/three.js-master/examples/jsm/libs/draco/"

        ); // Cambié la ruta de Draco para que sea correcta

*const* gltfLoader = new GLTFLoader();

        gltfLoader.setDRACOLoader(dracoLoader);

*const* gltf = await gltfLoader.loadAsync(

          "./js/three.js-master/examples/models/gltf/kira.glb"

        ); // Verifica la ruta correcta del modelo

        gltf.scene.traverse((*n*) *=>* {

          if (*n*.name === "head") OOI.head = *n*;

          if (*n*.name === "lowerarm\_l") OOI.lowerarm\_l = *n*;

          if (*n*.name === "Upperarm\_l") OOI.Upperarm\_l = *n*;

          if (*n*.name === "hand\_l") OOI.hand\_l = *n*;

          if (*n*.name === "target\_hand\_l") OOI.target\_hand\_l = *n*;

          if (*n*.name === "boule") OOI.sphere = *n*;

          if (*n*.name === "Kira\_Shirt\_left") OOI.kira = *n*;

        });

        scene.add(gltf.scene);

*const* targetPosition = OOI.sphere.position.clone(); // for orbit controls

        OOI.hand\_l.attach(OOI.sphere);

        // mirror sphere cube-camera

*const* cubeRenderTarget = new THREE.WebGLCubeRenderTarget(1024);

        mirrorSphereCamera = new THREE.CubeCamera(0.05, 50, cubeRenderTarget);

        scene.add(mirrorSphereCamera);

*const* mirrorSphereMaterial = new THREE.MeshBasicMaterial({

          envMap: cubeRenderTarget.texture,

        });

        OOI.sphere.material = mirrorSphereMaterial;

        OOI.kira.add(OOI.kira.skeleton.bones[0]);

*const* iks = [

          {

            target: 22, // "target\_hand\_l"

            effector: 6, // "hand\_l"

            links: [

              {

                index: 5, // "lowerarm\_l"

                rotationMin: new THREE.Vector3(1.2, -1.8, -0.4),

                rotationMax: new THREE.Vector3(1.7, -1.1, 0.3),

              },

              {

                index: 4, // "Upperarm\_l"

                rotationMin: new THREE.Vector3(0.1, -0.7, -1.8),

                rotationMax: new THREE.Vector3(1.1, 0, -1.4),

              },

            ],

          },

        ];

        IKSolver = new CCDIKSolver(OOI.kira, iks);

*const* ccdikhelper = new CCDIKHelper(OOI.kira, iks, 0.01);

        scene.add(ccdikhelper);

        gui = new GUI();

        gui.add(conf, "followSphere").name("follow sphere");

        gui.add(conf, "turnHead").name("turn head");

        gui.add(conf, "ik\_solver").name("IK auto update");

        gui.add(conf, "update").name("IK manual update()");

        gui.open();

        //

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

        renderer.setPixelRatio(window.devicePixelRatio);

        renderer.setSize(window.innerWidth, window.innerHeight);

        renderer.setAnimationLoop(animate);

        document.body.appendChild(renderer.domElement);

        //

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

        orbitControls.minDistance = 0.2;

        orbitControls.maxDistance = 1.5;

        orbitControls.enableDamping = true;

        orbitControls.target.copy(targetPosition);

        transformControls = new TransformControls(camera, renderer.domElement);

        transformControls.size = 0.75;

        transformControls.showX = false;

        transformControls.space = "world";

        transformControls.attach(OOI.target\_hand\_l);

        scene.add(transformControls);

        // disable orbitControls while using transformControls

        transformControls.addEventListener(

          "mouseDown",

          () *=>* (orbitControls.enabled = false)

        );

        transformControls.addEventListener(

          "mouseUp",

          () *=>* (orbitControls.enabled = true)

        );

        //

        stats = new Stats();

        document.body.appendChild(stats.dom);

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

      }

*function* animate() {

        if (OOI.sphere && mirrorSphereCamera) {

          OOI.sphere.visible = false;

          OOI.sphere.getWorldPosition(mirrorSphereCamera.position);

          mirrorSphereCamera.update(renderer, scene);

          OOI.sphere.visible = true;

        }

        if (OOI.sphere && conf.followSphere) {

          // orbitControls follows the sphere

          OOI.sphere.getWorldPosition(v0);

          orbitControls.target.lerp(v0, 0.1);

        }

        if (OOI.head && OOI.sphere && conf.turnHead) {

          // turn head

          OOI.sphere.getWorldPosition(v0);

          OOI.head.lookAt(v0);

          OOI.head.rotation.set(

            OOI.head.rotation.x,

            OOI.head.rotation.y + Math.PI,

            OOI.head.rotation.z

          );

        }

        if (conf.ik\_solver) {

          updateIK();

        }

        orbitControls.update();

        renderer.render(scene, camera);

        stats.update(); // fps stats

      }

*function* updateIK() {

        if (IKSolver) IKSolver.update();

        scene.traverse(*function* (*object*) {

          if (*object*.isSkinnedMesh) *object*.computeBoundingSphere();

        });

      }

*function* onWindowResize() {

        camera.aspect = window.innerWidth / window.innerHeight;

        camera.updateProjectionMatrix();

        renderer.setSize(window.innerWidth, window.innerHeight);

      }

    </script>

  </body>

</html>



