```
//#region IMPORT
import './style.css'
//import './checker.png'
import * as THREE from 'three'
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls.js'
import * as dat from 'dat.gui'
import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader.js'
import { FBXLoader } from 'three/examples/jsm/loaders/FBXLoader.js';
import { AnimationMixer, BlendingSrcFactor, DstAlphaFactor, MathUtils, Object3D, OneFactor,
PCFShadowMap, SkeletonHelper, SrcAlphaFactor, SubtractEquation, Vector3 } from 'three'
import Stats from 'three/examples/jsm/libs/stats.module.js';
import { DepthOfFieldEffect, SelectiveBloomEffect, BloomEffect, EffectComposer, EffectPass,
RenderPass } from "postprocessing";
import { Water } from 'three/examples/jsm/objects/water2.js';
//import { Water } from 'three/examples/jsm/objects/Water.js';
import { Refractor } from 'three/examples/jsm/objects/Refractor.js';
import { Reflector } from 'three/examples/jsm/objects/Reflector.js';
/* import { EffectComposer } from 'three/examples/jsm/postprocessing/EffectComposer.js';
import { RenderPass } from 'three/examples/jsm/postprocessing/RenderPass.js';
import { UnrealBloomPass } from 'three/examples/jsm/postprocessing/UnrealBloomPass.js';
 */
import { gsap } from 'gsap/all';
//import { CustomEase } from "gsap/CustomEase";
import { radToDeg } from 'three/src/math/MathUtils'
//#endregion

//#region SCENE VARIABLES

// Debug
const gui = new dat.GUI();
gui.close()
const stats = Stats();
stats.showPanel(0) // 0: fps, 1: ms, 2: mb, 3+: custom
//console.log(stats.dom.outerHTML);
//document.body.appendChild(stats.dom);
```

```
// Clock
const clock = new THREE.Clock();

// Texture Loader
const txtLoader = new THREE.TextureLoader();

// GLTF Loader
const gltfLoader = new GLTFLoader();
const fbxLoader = new FBXLoader();

// Canvas
const canvas = document.querySelector('canvas.webgl');

// Scene
const scene = new THREE.Scene();
let sceneBG = txtLoader.load("textures/sceneBG.jpg");
sceneBG.mapping = THREE.EquirectangularReflectionMapping;
scene.background = sceneBG;

// Uniforms
let uniforms = {
    time: { value: 1.0 },
    resolution: { type: "v2", value: new THREE.Vector2() },
    sampleTxt: {
        value: txtLoader.load('https://threejsfundamentals.org/threejs/resources/images/
checker.png')
    }
};

window.onload = function () {
    let val = document.getElementById(`loadingNum`);
    gsap.to({}, {
        duration: 8, onUpdate: function () {
            val.innerHTML = round(this.progress()) * 100;
        }, onComplete: function () {
            let startButton = document.querySelector('#splash button');
```

```
            startButton.classList.remove(`deactivated`);
        }
    })
}

//#region Click Listeners
let cursor = document.getElementById('cursor');
cursor.addEventListener("click", advanceScene);

let startButton = document.querySelector('#splash button');
startButton.addEventListener("click", function () {
    console.log('%c starting 3D experience', 'color: lightgreen');
    canBegin = true;
    windVideo.play();
    gsap.to(document.getElementById('splash'), {
        duration: 2, ease: "power2.inOut", opacity: 0, onComplete: function () {
            document.getElementById('splash').style.display = 'none';
        }
    });
});

let previousButton = document.getElementById('previous');
previousButton.addEventListener("click", function () {
    console.log('%c going to previous page', 'color: lightgreen');
    previousScene();
});
let skipButton = document.getElementById('skip');
skipButton.addEventListener("click", function () {
    console.log('%c skipping to next page', 'color: lightgreen');
    advanceScene();
});
let restartButton = document.getElementById('restart');
restartButton.addEventListener("click", function () {
    console.log('%c restarting book', 'color: lightgreen');
    restartScene();
});
```

```
let autoplayButton = document.getElementById('autoplay');
autoplayButton.addEventListener("click", function () {
    gsapT1.to({}, {
        duration: 1,
        onComplete: function () {
            autoplay.setEnabled(!autoplay._enabled)
            if (cursor.style.display == 'block') {
                advanceScene();
            }
        }
    })
});

let part1Button = document.getElementById('p1');
part1Button.addEventListener("click", function () {
    console.log('%c going part 1', 'color: lightgreen');
    part1Button.classList.remove('active');
    part2Button.classList.remove('active');
    part3Button.classList.remove('active');

    part1Button.classList.add('active');
    goToScene(0);
});
let part2Button = document.getElementById('p2');
part2Button.addEventListener("click", function () {
    console.log('%c going part 2', 'color: lightgreen');
    part1Button.classList.remove('active');
    part2Button.classList.remove('active');
    part3Button.classList.remove('active');

    part2Button.classList.add('active');
    goToScene(4);
});
let part3Button = document.getElementById('p3');
part3Button.addEventListener("click", function () {
    console.log('%c going part 3', 'color: lightgreen');
```

```
    part1Button.classList.remove('active');
    part2Button.classList.remove('active');
    part3Button.classList.remove('active');

    part3Button.classList.add('active');
    goToScene(12);
});
//#endregion

//#endregion

//#region OBJECT VARIBLES
/**
 * GLOBAL VARIABLES
 */

// init objects
//const bloomLayer = new THREE.Layers();

let effectComposer, renderPass, bloomPass, DOFPass;
let axesHelper;

let boyAnimParams;
let mats = [];
let gltfModels = [];
let mixers = [];

let windVideo;
let boxMesh, sphereMesh, pointsMesh, water;
let boyMixer, girlMixer, heartMixer, birdMixer, bird2Mixer, bird3Mixer, fishMixer, treeMixer;
let boyAnimations, girlAnimations, heartAnimations, birdAnimations, bird2Animations,
bird3Animations, fishAnimations;
let boySkeleton, birdSkeleton, boyModel, girlModel, domeModel, sandWispModel, windWispModel;
let flowerModel, heartModel, heartPointsModel, birdModel, bird2Model, bird3Model, fishModel,
treeModel;
let pole_walking_NLA, sitting_NLA, start_walking_NLA, movePos1_NLA, walk_cycle_NLA;
```

```
let blow_kiss_NLA, spin_NLA;

let allNarration, activeSceneNum = 0;
let fadeOverride = false;

function timelineObj(name, repeat, actors, readyPositions, playTransition, playActions) {
    this.name = name;
    this.repeat = repeat;
    this.actors = actors;
    this.readyPositions = readyPositions;
    this.playTransition = playTransition;
    this.playActions = playActions;
}
const timelineClips = [];
// T1 — handles finite scene/mesh animations
// T1_2 — handles repeating scene/mesh animations
// T2 — handles HTML transitions
// T3 — handles material transitions
let gsapT1 = gsap.timeline({ repeat: 0 });
let gsapT1_2 = gsap.timeline({ repeat: -1 });
let gsapT2 = gsap.timeline({ repeat: 0 });
let gsapT3 = gsap.timeline({ repeat: 0 });

let controls, camera, renderer;
let meshLoaded = false, mixerLoaded = false, isRotating = true, canBegin = false;
// !
let autoRot = false;

let rotObj = new Object3D();
scene.add(rotObj);

// !
function autoplayObj() {
    this._enabled = false;
    this.lastMouseTime = 0;
    this.delta = clock.getElapsedTime() - this.lastMouseTime;
```

```
    this.setEnabled = function (bool) {
        this._enabled = bool;
        console.log(`%c autoplay: ${autoplay._enabled}`, 'color: lightgreen');

    }

    this.tickAP = function () {
        // enable if no mouse movement for 30s
        // disable if any mouse movement
        /* console.log(`last mouse time: ${this.lastMouseTime}`)
        console.log(`enabled: ${this._enabled}`)
        console.log(`delta: ${this.delta}`) */
        if (!this._enabled) {
            this.delta = clock.getElapsedTime() - this.lastMouseTime;
            if (this.delta > 60) {
                this.setEnabled(true);
                console.log(`enabling autoplay`)
            }
        }
    }
}

let autoplay = new autoplayObj();
//#endregion

/**
 * INIT OJBECTS
 */
function initObjects() {
    //#region GEO/TXT
    // Geometry
    const boxGeo = new THREE.BoxGeometry(1, 1, 1);
    const sphereGeo = new THREE.SphereGeometry(.5, 100, 100);
    const pointsGeo = new THREE.SphereGeometry(1, 10, 10);
```

```
    //  Textures
    const checkTxt = txtLoader.load(`https://threejsfundamentals.org/threejs/resources/images/
checker.png`);
    checkTxt.wrapS = THREE.RepeatWrapping;
    checkTxt.wrapT = THREE.RepeatWrapping;
    checkTxt.repeat.set(10, 10);
    checkTxt.magFilter = THREE.NearestFilter;

    const wispTxt = txtLoader.load(`SandWispTxt2k.png`);
    const wispTxtAlpha = txtLoader.load(`SandWispTxt2k_alpha.png`);

    windVideo = document.getElementById('windVideo');
    const windTxt = new THREE.VideoTexture(windVideo);
    //#endregion

    //#region MATERIALS
    const whiteMat = new THREE.MeshBasicMaterial();
    whiteMat.color = new THREE.Color(0xfefefe);

    const phongMat = new THREE.MeshPhongMaterial({
        color: 0xffffff,
        side: THREE.DoubleSide
    });
    phongMat.name = `phong mat`;
    mats.push(phongMat);

    const checkMat = new THREE.MeshBasicMaterial({
        map: checkTxt,
        side: THREE.DoubleSide
    });

    const wireMat = new THREE.MeshPhongMaterial({
        wireframe: true,
        opacity: .5,
        transparent: true,
    });
```

```
wireMat.name = `wireframe mat`;
mats.push(wireMat);

const rockMat = new THREE.MeshStandardMaterial({
    wireframe: false,
    opacity: 1,
    transparent: true,
    map: txtLoader.load(`rock_diffuse.jpg`),
});
rockMat.name = `rock diffuse mat`;
mats.push(rockMat);

// ! fix depthtest and render order of snad wisp and dome
const sandWispMat = new THREE.MeshBasicMaterial({
    map: wispTxt,
    //color: 0xffffff,
    side: THREE.DoubleSide,
    alphaMap: wispTxtAlpha,
    transparent: true,
})
sandWispMat.name = `sand wisp mat`;
mats.push(sandWispMat);

const windMat = new THREE.MeshBasicMaterial({
    transparent: true,
    map: windTxt,
    side: THREE.DoubleSide,
    opacity: .25,
    alphaMap: windTxt,
});
windMat.name = `wind video mat`
mats.push(windMat);

/* const domeMat = new THREE.MeshBasicMaterial({
    //transparent: true,
    map: txtLoader.load("textures/dome_1.jpg"),
```

```
    side: THREE.BackSide,
});
domeMat.name = `dome mat`
mats.push(domeMat); */

const treeLeavesMat = new THREE.MeshStandardMaterial({
    color: 0xc09253,
    alphaMap: txtLoader.load(`tree/m_leaves_alpha.png`),
    transparent: true,
});
treeLeavesMat.name = `tree leaves mat`;
mats.push(treeLeavesMat);

const flowerMat = new THREE.MeshBasicMaterial({
    map: txtLoader.load("flower_diffuse.png"),
    side: THREE.DoubleSide,
    transparent: true,
});
flowerMat.name = `flower mat`;
mats.push(flowerMat);

/* const pointsMat = new THREE.PointsMaterial({
    transparent: true,
    size: .0005,
}) */

/* const pointsWindMat = new THREE.PointsMaterial({
    transparent: true,
    //map: windTxt,
    //color: 0xc09253,
    side: THREE.DoubleSide,
    //alphaMap: windTxt,
    size: .05,
}); */

const glowMat = new THREE.MeshStandardMaterial({
```

```
    color: 0xffffff,
    //emissive: 0xffffff,
    //emissiveIntensity: 10,
    transparent: true,
    skinning: true,
});
glowMat.name = `glow mat`;
mats.push(glowMat);

// Shader Materials
//#region SHADERMATS
const shaderMat = new THREE.ShaderMaterial({
    uniforms: uniforms,
    vertexShader: document.getElementById('vertexShader').textContent,
    fragmentShader: document.getElementById('fragmentShader').textContent,
    side: THREE.DoubleSide,
    blending: THREE.CustomBlending,
    blendSrc: SrcAlphaFactor,
    blendDst: DstAlphaFactor,
});

const shaderMat2 = new THREE.ShaderMaterial({
    uniforms: uniforms,
    vertexShader: document.getElementById('vertexShader').textContent,
    fragmentShader: document.getElementById('fragmentShader2').textContent,
    side: THREE.DoubleSide
});

const shaderMat3 = new THREE.ShaderMaterial({
    uniforms: uniforms,
    vertexShader: document.getElementById('vertexShader').textContent,
    fragmentShader: document.getElementById('fragmentShader3').textContent,
    side: THREE.DoubleSide
});

const wireFrontMat = new THREE.ShaderMaterial({
```

```
    vertexShader: document.getElementById('vertexShader').textContent,
    fragmentShader: document.getElementById('fragmentShader4').textContent,
    side: THREE.DoubleSide,
});
//#endregion

//#endregion

//#region MESH

sphereMesh = new THREE.Mesh(sphereGeo, glowMat);
sphereMesh.scale.set(1, 1, 1);
sphereMesh.position.set(.5, -1, -2);
scene.add(sphereMesh);

boxMesh = new THREE.Mesh(boxGeo, glowMat);
boxMesh.scale.set(1, 1, 1);
boxMesh.position.set(-1, 1, -1);
scene.add(boxMesh);

/* pointsMesh = new THREE.Points(pointsGeo, pointsMat);
scene.add(pointsMesh); */

//#endregion

//#region GLTF

/* gltfLoader.load(`dome_v1.glb`, (gltf) => {
    domeModel = gltf.scene;

    //set transforms
    let scale = 60;
    domeModel.scale.set(scale, scale, scale);
    domeModel.position.set(5, scale / 3.5, 5);

    //console.log(wispModel)
```

```
    // assign material and shadow
    domeModel.traverse(function (child) {
        if (child.isMesh) {
            //object.receiveShadow = true;
            child.material = domeMat;
        }
    });



    // add model to scene
    gltfModels.push(domeModel);
    //scene.add(domeModel);
}); */



/**
 * LOAD WISPS GLTF
 */
gltfLoader.load(`sandWisp_v1.gltf`, (gltf) => {
    sandWispModel = gltf.scene;

    //set transforms
    sandWispModel.scale.set(2, 2, 2);
    sandWispModel.position.set(-1, 0, -1);
    //console.log(wispModel)

    // assign material and shadow
    sandWispModel.traverse(function (child) {
        if (child.isMesh) {
            //object.receiveShadow = true;
            child.material = sandWispMat;
        }
    });

    // add model to scene
```

```
    gltfModels.push(sandWispModel);
    scene.add(sandWispModel);
});

gltfLoader.load(`wind_wisp_long1.gltf`, (gltf) => {
    windWispModel = gltf.scene;

    //set transforms
    windWispModel.scale.set(1, 1, 1);
    windWispModel.position.set(0, 0, -2);
    //console.log(`%c ${windWispModel}`, `color: #e26f03`)

    //let geometries = [];

    // assign material and shadow
    windWispModel.traverse(function (child) {
        if (child.isMesh) {
            //object.receiveShadow = true;
            child.material = windMat;
            //geometries.push(child.geometry);
        }
    });
    //console.log(geometries);



    // add model to scene
    gltfModels.push(windWispModel);
    scene.add(windWispModel);

    // add points model to scene
    //scene.add(new THREE.Points(geometries[0], pointsWindMat));
});

/**
 * LOAD FLOWER GLTF
 */
```

```
gltfLoader.load(`desert_flower_v4.gltf`, (gltf) => {
    flowerModel = gltf.scene;

    //set transforms
    flowerModel.scale.set(.1, .1, .1);

    // assign material and shadow
    flowerModel.traverse(function (child) {
        if (child.isMesh) {
            if (child.name.includes("flower")) {
                //console.log("child flower name: " + child.name);
                child.material = flowerMat;
            } else {
                child.material = rockMat;
            }
        }
    });

    // add model to scene
    gltfModels.push(flowerModel);
    scene.add(flowerModel);
});

/**
* LOAD HEART GLTF
*/
gltfLoader.load(`crystal_heart.gltf`, (gltf) => {
    heartModel = gltf.scene;
    heartModel.name = `heart`;

    //set transforms
    heartModel.scale.set(1, 1, 1);

    //heartPointsModel = new THREE.Points(heartModel, pointsMat);

    // assign material and shadow
```

```
    /* heartModel.traverse(function (child) {
        if (child.isMesh) {
            child.material = pointsMat;
        }
    }); */

    // add model to scene
    //gltfModels.push(heartModel);
    gltfModels.push(heartModel);
    scene.add(heartModel);

    // init animation mixer
    heartMixer = new THREE.AnimationMixer(heartModel);
    heartAnimations = gltf.animations;
    mixers.push(heartMixer);

    spin_NLA = heartMixer.clipAction(gltf.animations[0]);
    spin_NLA.play();
});

gltfLoader.load(`tree/tree_v5.glb`, (gltf) => {
    treeModel = gltf.scene;
    treeModel.name = `tree`;

    //set transforms
    treeModel.scale.set(.25, .25, .25);
    treeModel.position.set(0, -.5, .25);
    treeModel.rotation.set(0, degToRad(91), 0);

    // assign material and shadow
    treeModel.traverse(function (child) {
        if (child.isMesh) {
            if (child.name.includes(`leaf`)) {
                child.material = treeLeavesMat;
            } else {
                child.material = glowMat;
```

```
            }
        }
    });

    // add model to scene
    //gltfModels.push(treeModel);
    gltfModels.push(treeModel);
    scene.add(treeModel);
    //console.log(treeModel)

    // init animation mixer
    treeMixer = new THREE.AnimationMixer(treeModel);
    treeModel.animations = gltf.animations;
    mixers.push(treeMixer);

    //treeMixer.clipAction(gltf.animations[0]).play();
});

/**
* LOAD BIRDIE GLTF
*/
fbxLoader.load(`birdie_v8.fbx`, function (fbx) {
    birdModel = fbx;
    birdModel.name = `bird`;

    //set transforms
    birdModel.scale.set(.0001, .0001, .0001);

    // assign material and shadow
    /* birdModel.traverse(function (child) {
        if (child.isMesh) {
            child.material = glowMat;
        }
    }); */

    // add model to scene
```

```
    gltfModels.push(birdModel);
    scene.add(birdModel);

    // init animation mixer
    birdMixer = new THREE.AnimationMixer(birdModel);
    birdAnimations = fbx.animations;
    mixers.push(birdMixer);

    birdModel.activeClip = birdMixer.clipAction(fbx.animations[3]);
    birdModel.activeClip.play();
    //let fly_NLA = birdMixer.clipAction(fbx.animations[3]);
    //fly_NLA.play();

    //console.log(fbx.animations);

    //switchGLTFAnims(birdModel, birdMixer.clipAction(fbx.animations[0]))
});

fbxLoader.load(`birdie_v8.fbx`, function (fbx) {
    bird2Model = fbx;
    bird2Model.name = `bird2`;

    //set transforms
    bird2Model.scale.set(.0001, .0001, .0001);
    bird2Model.position.set(1, 3, -10);

    // assign material and shadow
    /* bird2Model.traverse(function (child) {
        if (child.isMesh) {
            child.material = glowMat;
        }
    }); */

    // add model to scene
    gltfModels.push(bird2Model);
    scene.add(bird2Model);
```

```
    // init animation mixer
    bird2Mixer = new THREE.AnimationMixer(bird2Model);
    bird2Animations = fbx.animations;
    mixers.push(bird2Mixer);

    bird2Model.activeClip = bird2Mixer.clipAction(fbx.animations[3]);
    bird2Model.activeClip.play();

    //console.log(fbx.animations);

    //switchGLTFAnims(bird2Model, bird2Mixer.clipAction(fbx.animations[0]))
});

gltfLoader.load(`birdie_8.glb`, function (gltf) {
    bird3Model = gltf.scene;
    bird3Model.name = `bird3 GLTF`;

    //set transforms
    bird3Model.scale.set(.01, .01, .01);

    // assign material and shadow
    bird3Model.traverse(function (child) {
        if (child.isMesh) {
            child.material = glowMat;
        }
    });

    // add model to scene
    gltfModels.push(bird3Model);
    scene.add(bird3Model);

    // init animation mixer
    bird3Mixer = new THREE.AnimationMixer(bird3Model);
    bird3Animations = gltf.animations;
    mixers.push(bird3Mixer);
```

```
    bird3Model.activeClip = bird3Mixer.clipAction(gltf.animations[0]);
    bird3Model.activeClip.play();
});

gltfLoader.load(`fibshOver_v2.glb`, function (glb) {
    fishModel = glb.scene;
    fishModel.name = `fibsh`;

    //set transforms
    fishModel.scale.set(.1, .1, .1);
    fishModel.position.set(0, .25, -1);
    fishModel.rotation.set(0, degToRad(180), 0);

    // assign material and shadow
    fishModel.traverse(function (child) {
        if (child.isMesh) {
            child.material = glowMat;
        }
    });

    // add model to scene
    gltfModels.push(fishModel);
    scene.add(fishModel);

    // init animation mixer
    fishMixer = new THREE.AnimationMixer(fishModel);
    fishAnimations = glb.animations;
    mixers.push(fishMixer);
    // 4 and 6
    fishModel.activeClip = fishMixer.clipAction(glb.animations[0]);
    fishModel.activeClip.play();
    fishModel.activeClip.paused = true;

    /* var action = fishMixer.clipAction(glb.animations[0]);
    action.loop = THREE.LoopOnce; */
```

```
    //console.log(glb.animations);)
});

/**
* LOAD BOY GLTF
*/

gltfLoader.load(`boy_v16.glb`, (gltf) => {
    boyModel = gltf.scene;
    boyModel.name = `boy`;

    //set transforms
    boyModel.scale.set(.1, .1, .1);

    // assign cast shadow and materials
    /* boyModel.traverse(function (child) {
        if (child.isMesh) {
            //object.castShadow = true;
            //object.receiveShadow = true;
            //console.log("object name: " + object.name)
            child.material = glowMat;
        }
    }); */

    // add BOY to scene
    gltfModels.push(boyModel);
    scene.add(boyModel);
    //console.log(boyModel)
    //console.log(gltfModels)

    // show rig skeleton
    /* boySkeleton = new THREE.SkeletonHelper(boyModel);
    boySkeleton.visible = true; */
    //scene.add(skeleton);
```

```
    // init animation mixer
    boyMixer = new THREE.AnimationMixer(boyModel);
    boyAnimations = gltf.animations;
    mixers.push(boyMixer);

    movePos1_NLA = boyMixer.clipAction(gltf.animations[0]);
    pole_walking_NLA = boyMixer.clipAction(gltf.animations[1]);
    sitting_NLA = boyMixer.clipAction(gltf.animations[2]);
    start_walking_NLA = boyMixer.clipAction(gltf.animations[3]);
    walk_cycle_NLA = boyMixer.clipAction(gltf.animations[4]);

    boyModel.activeClip = walk_cycle_NLA;
    boyModel.activeClip.play();

    //#region BOY GUI
    const folder1 = gui.addFolder('boy controls');

    boyAnimParams = {
        'sit down': function () { switchGLTFAnims(boyModel, sitting_NLA) },
        'walk cycle': function () { switchGLTFAnims(boyModel, walk_cycle_NLA) },
        'move position 1': function () { switchGLTFAnims(boyModel, movePos1_NLA) },
        'pole walking': function () { switchGLTFAnims(boyModel, pole_walking_NLA) },
        'start walking': function () { switchGLTFAnims(boyModel, start_walking_NLA) }
    }
    folder1.add(boyAnimParams, 'sit down');
    folder1.add(boyAnimParams, 'walk cycle');
    folder1.add(boyAnimParams, 'move position 1');
    folder1.add(boyAnimParams, 'pole walking');
    folder1.add(boyAnimParams, 'start walking');

    //#endregion
});

/**
* LOAD GIRL GLTF
*/
```

```
gltfLoader.load(`theGirl_v6.gltf`, (gltf) => {
    girlModel = gltf.scene;

    //set transforms
    girlModel.scale.set(.1, .1, .1);
    girlModel.position.set(-.6, 0, 2);
    girlModel.rotation.set(0, degToRad(170), 0);

    // assign cast shadow and materials
    /* girlModel.traverse(function (child) {
        if (child.isMesh) {
            //object.castShadow = true;
            child.material = txtMat;
        }
    }); */

    // add GIRL to scene
    gltfModels.push(girlModel);
    scene.add(girlModel);
    /* console.log(`girlModel: `);
    console.log(girlModel)
    console.log(`gltfModels[]: `);
    console.log(gltfModels) */

    // init animation mixer
    girlMixer = new THREE.AnimationMixer(girlModel);
    girlAnimations = gltf.animations;
    mixers.push(girlMixer);

    blow_kiss_NLA = girlMixer.clipAction(gltf.animations[0]);
    blow_kiss_NLA.play();
});
//#endregion

//#region WATER INIT
```

```
    const waterGeometry = new THREE.PlaneGeometry(20, 20);
    const flowMap = txtLoader.load('textures/water/Water_1_M_Flow.jpg');

    // water.jsm
    water = new Water(waterGeometry, {
        scale: 0,
        textureWidth: 1024,
        textureHeight: 1024,
        flowMap: flowMap,
    });

    water.position.y = .5;
    water.rotation.x = degToRad(270);

    scene.add(water);
    //#endregion

    //#region LIGHTS

    const hemiLight = new THREE.HemisphereLight(0xffffff, 0x444444);
    hemiLight.position.set(0, 20, 0);
    hemiLight.layers.enableAll();
    scene.add(hemiLight);

    //#endregion
}
function switchGLTFAnims(model, newClip) {
    if (model.activeClip != newClip) {
        console.log(`%c ${model.name}: ${newClip._clip.name}`, `color: DarkGoldenRod`);
        newClip.enabled = true;
        newClip.setEffectiveWeight(1);
        newClip.reset();
        newClip.play();
        model.activeClip.crossFadeTo(newClip, .5, false);

        model.activeClip = newClip;
```

```
    }
}

/**
 * INIT SCENE
 */
function initScene() {
    //#region LISTENERS/SIZE
    /**
     * Sizes
     */
    const sizes = {
        width: window.innerWidth,
        height: window.innerHeight
    }

    axesHelper = new THREE.AxesHelper(5);
    //scene.add(axesHelper);

    window.addEventListener('resize', () => {
        // Update sizes
        sizes.width = window.innerWidth
        sizes.height = window.innerHeight

        // Update camera
        camera.aspect = sizes.width / sizes.height
        camera.updateProjectionMatrix()

        // Update renderer
        renderer.setSize(sizes.width, sizes.height)
        renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2))
        effectComposer.setSize(sizes.width, sizes.height)

        bloomPass.blurPass.width = sizes.width;
        bloomPass.blurPass.height = sizes.height;
    })
```

```
//#endregion

/**
 * Camera
 */
camera = new THREE.PerspectiveCamera(75, sizes.width / sizes.height, 0.1, 100)
camera.position.x = 0
camera.position.y = 0
camera.position.z = 2
scene.add(camera)

const folder2 = gui.addFolder('camera controls');

folder2.add(camera.position, "x").min(-10).max(10);
folder2.add(camera.position, "y").min(-10).max(10);
folder2.add(camera.position, "z").min(-10).max(10);

/* folder2.add(camera.rotation, "x").min(degToRad(-1)).max(degToRad(1));
folder2.add(camera.rotation, "y").min(degToRad(-1)).max(degToRad(1));
folder2.add(camera.rotation, "z").min(degToRad(-1)).max(degToRad(1)); */

// Controls
controls = new OrbitControls(camera, canvas)
controls.enableDamping = true;
//controls.enabled = false;

/**
 * Renderer
 */
renderer = new THREE.WebGLRenderer({
    canvas: canvas
})
renderer.setSize(sizes.width, sizes.height)
renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2))
renderer.shadowMap.enabled = true;
```

```
    renderer.shadowMap.type = PCFShadowMap;

    // Postprocessing
    //#region BLOOM



    effectComposer = new EffectComposer(renderer);
    renderPass = new RenderPass(scene, camera);

    //bloomPass = new UnrealBloomPass(new THREE.Vector2(window.innerWidth, window.innerHeight),
1.5, 0.4, 0.95);
    bloomPass = new BloomEffect({
        intensity: 2,
        luminanceThreshold: .5
    })
    let dofFocus = new THREE.Vector2(.15, .025);
    DOFPass = new DepthOfFieldEffect(camera, {
        focusDistance: 0.15,
        focalLength: 0.025,
        bokehScale: 4.0,
        //height: 480,
        //target: dofTarget,
    })
    const effectPass = new EffectPass(
        camera,
        bloomPass,
        DOFPass,
    );

    effectComposer.addPass(renderPass);
    //effectComposer.addPass(bloomPass);
    effectComposer.addPass(effectPass);

    //#endregion

    // Load GUI Items
```

```
    const folder3 = gui.addFolder('bloom controls');
    folder3.add(bloomPass.blurPass, "scale").min(0).max(50);
    folder3.add(bloomPass.blurPass, "width").min(0).max(1080);
    folder3.add(bloomPass.blurPass, "height").min(-25).max(1080);
    folder3.add(bloomPass, 'intensity').min(-25).max(100);

    const folder4 = gui.addFolder('DOF controls');
    folder4.add(DOFPass, "bokehScale").min(0).max(8);
    folder4.add(dofFocus, "x", 0.0, 1.0, 0.001).onChange((value) => {
        DOFPass.circleOfConfusionMaterial.uniforms.focusDistance.value = value;
    });
    folder4.add(dofFocus, "y", 0.0, 1.0, 0.001).onChange((value) => {
        DOFPass.circleOfConfusionMaterial.uniforms.focalLength.value = value;
    });

    setDOFDistance(new THREE.Vector3(0, 0, 0), .25);

    /* folder4.add(dofTarget, "x").min(-10).max(10).onChange(function () {
        DOFPass.setTarget(dofTarget);
    });
    folder4.add(dofTarget, "y").min(-10).max(10);
    folder4.add(dofTarget, "z").min(-10).max(10); */
}

function setDOFDistance(vec3, dur) {
    let dist = DOFPass.calculateFocusDistance(vec3);
    //console.log(dist);

    let cocFocDist = DOFPass.circleOfConfusionMaterial.uniforms.focusDistance;
    if (dur == 0) {
        DOFPass.circleOfConfusionMaterial.uniforms.focusDistance.value = dist;
    } else {
        gsap.to(cocFocDist, { duration: dur, value: dist });
    }
}
```

```
/**
 * INIT TIMELINE
 */

//#region GSAP ANIMS
function initTimeline() {
    //#region TIMELINE OBJs
    timelineClips.push(
        new timelineObj(
            'zoom through rocks to flower', 0,
            [flowerModel, boyModel, sandWispModel],
            //#region anims
            // ready positions
            function () {
                // cycle through all actors and set their inital positions
                scene.background = sceneBG;
                flowerModel.position.set(0, 0, 0);
                boyModel.position.set(0, 0, 0);
                sandWispModel.position.set(-1, 0, -1);

                //camera.add(domeModel);

                switchGLTFAnims(boyModel, sitting_NLA);

                //gsapT1.to(camera.position, { duration: .1, z: -1.75, y: .35, x: -.75 });
                camera.position.set(-.75, .35, -1.75);
                setDOFDistance(new THREE.Vector3(0, 0, 0), .25);

                let pos;
                flowerModel.traverse(function (child) {
                    if (child.name.includes("flower")) {
                        pos = child.position;
                        camera.lookAt(pos);
                    }
                })
```

```
            bloomPass.intensity = 2;
            bloomPass.luminanceThreshold = .5;
            bloomPass.blurPass.scale = 1;
            bloomPass.blurPass.width = window.innerWidth;
            bloomPass.blurPass.height = window.innerHeight;
        },
        // play transition
        function () {
            fadeMats(mats, gltfModels, 1, .75);
        },
        // play actions
        function () {
            gsapT1.clear();

            gsapT1.to({}, { duration: 2, }); //stall for 2s

            //to camera pos
            gsapT1.to(camera.position, { duration: 10, ease: "power2.inOut", z: 21.75 },);
            gsapT1.to(camera.position, {
                duration: 10, ease: "power2.inOut", x: -1.15, onComplete: function () {
                    flowerModel.traverse(function (child) {
                        if (child.name.includes("flower")) {
                            let pos = child.position;
                            DOFPass.circleOfConfusionMaterial.uniforms.focusDistance.value =
.004;

                        }
                    })
                }
            }, `<`)

            // false = fade out | true = custom transition handler
            //gsapT1.call(function () { fadeOverride = true });

            /* // !
            gsapT1.call(function () {
                _actionsComplete = true;
```

```
        }); */
    }
    //#endregion
),
new timelineObj(
    'obscure flower and fade in boy', 0,
    [flowerModel, boyModel, sandWispModel, axesHelper],
    //#region anims
    // ready positions
    function () {
        flowerModel.position.set(0, 0, 0);
        boyModel.position.set(0, 0, 0);
        sandWispModel.position.set(-1, 0, -1);

        switchGLTFAnims(boyModel, sitting_NLA);
        boyMixer.clipAction(boyAnimations[5]).play();


        camera.position.set(-.25, .7, 2);
        setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
        //camera.lookAt(new THREE.Vector3(20, 20, 20));
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, 1);
    },
    // play actions
    function () {
        gsapT1.clear();

        /* mats.forEach(mat => {
            gsapT1.to(mat, { duration: .5, opacity: 0 }, '<');
        }); */

        // lerp lookat() function
        let vecFrom = camera.getWorldDirection(new THREE.Vector3());
```

```
        let vecTo = new THREE.Vector3(1.35, 0, 0);
        let state;
        gsapT1.to({}, {
            duration: 2, ease: "power2.inOut",
            onUpdate: function () {
                state = vecFrom.lerp(vecTo, this.progress());
                camera.lookAt(state);
                //console.log(`lookat ` + this.progress());
            }
        }, `<`);

        gsapT1.to(mats[0], { duration: 1, opacity: 0 });

        mats.forEach(mat => {
            gsapT1.to(mat, { duration: 1, opacity: 1 }, '<');
        });
    }
    //#endregion
),
new timelineObj(
    'fade girl and blow kiss', -1,
    [boyModel, girlModel, heartModel],
    //#region anims
    // ready positions
    function () {
        boyModel.position.set(0, 0, 0);
        girlModel.position.set(-.6, 0, 2);
        heartModel.position.set(-.6, .5, 1.85);
        heartModel.scale.set(.1, .1, .1),

            switchGLTFAnims(boyModel, sitting_NLA);

        camera.position.set(-2, .25, .6);
        camera.lookAt(new THREE.Vector3(0, .5, 1));
        setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
```

```
    // fade out girl
    girlModel.traverse(child => {
        if (child.material) {
            child.material.transparent = true;
            child.material.opacity = .75;
        };
        //console.log(`%c FADE OUT GIRL SCENE 3`, 'color: #00FFE3')
    });
},
// play transition
function () {
    fadeMats(mats, [boyModel], 1, 1);
},
function () {
    gsapT1.clear();

    gsapT1.addLabel(`girlFadeIn`, `0`)
    gsapT1.addLabel(`heartFadeOut`, `+=6`)
    // fade girl in
    girlModel.traverse(child => {
        if (child.material) {
            child.material.transparent = true;
            gsapT1.to(child.material, { duration: 3.5, opacity: .75 }, 'girlFadeIn');
        };
        //console.log(`%c FADE OUT GIRL SCENE 3`, 'color: #00FFE3')
    });

    gsapT1.to(heartModel.position, { duration: 6, y: .25, z: .25 }, 'girlFadeIn');
    // fade heart in
    heartModel.traverse(child => {
        if (child.material) {
            child.material.transparent = true;
            gsapT1.to(child.material, { duration: .5, opacity: 1 }, 'girlFadeIn');
        };
    });
    heartModel.traverse(child => {
```

```
            if (child.material) {
                child.material.transparent = true;
                gsapT1.to(child.material, { duration: .25, opacity: 0 }, 'heartFadeOut');
            };
        });
    }
    //#endregion
),
new timelineObj(
    'boy turns into wind', 0,
    [windWispModel, boyModel],
    //#region anims
    // ready positions
    function () {
        boyModel.position.set(0, 0, 0);
        windWispModel.position.set(0, 0, -2);

        switchGLTFAnims(boyModel, sitting_NLA);

        //camera.position.set(-2.25, 0, .5);
        camera.position.set(-1.25, .5, 2.25);
        camera.lookAt(new THREE.Vector3(0, .5, .5));
        setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, 1.25);
    },
    // play actions
    function () {
        gsapT1.clear();
    }
    //#endregion
),
new timelineObj(
    'heart fades in and rotates', 0,
```

```
    [heartModel],
    //#region anims
    // ready positions
    function () {
        heartModel.position.set(0, 0, 0);
        heartModel.scale.set(1, 1, 1);

        camera.position.set(0, .55, -2);
        camera.lookAt(new THREE.Vector3(0, .5, 0));
        setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, 1.25);
    },
    // play actions
    function () {
        gsapT1.clear();
        gsapT1.to({}, { duration: 1, });

        //heart fade in and rotate

    }
    //#endregion
),
new timelineObj(
    'bird takes off', 0,
    [bird3Model],
    //#region anims
    // ready positions
    function () {
        bird3Model.position.set(0, 0, 0);
        bird3Model.rotation.set(0, 0, 0);

        //switchGLTFAnims(bird3Model, bird3Mixer.clipAction(bird3Animations[0]))
        bird3Mixer.setTime(0);
```

```
        camera.position.set(-.5, .1, 1);
        camera.lookAt(new THREE.Vector3(.5, 0, 0));
        setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, 1.25);
    },
    // play actions
    function () {
        gsapT1.clear();
        gsapT1_2.clear();

        gsapT1.to({}, { duration: 1, }).call(function () {
            bird3Mixer.setTime(0);
            var action = bird3Mixer.clipAction(bird3Animations[0]);
            action.reset();
            action.loop = THREE.LoopOnce;
            action.clampWhenFinished = true;
            action.clamp
            action.play();
        });

        // false = fade out | true = custom transition handler
        //gsapT1.call(function () { fadeOverride = true });
    }
    //#endregion
),
new timelineObj(
    'camera pans around bird', 0,
    [birdModel, flowerModel],
    //#region anims
    // ready positions
    function () {
        birdModel.position.set(0, 0, 0);
```

```
            birdModel.rotation.set(0, 0, 0);

            rotObj.rotation.set(0, 0, 0);
            rotObj.position.set(0, 0, 0);

            flowerModel.position.set(0, -10, 0);
            flowerModel.traverse(child => {
                if (child.material) {
                    child.material.transparent = true;
                    child.material.opacity = 0;
                };
            });

            switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[2]))

            camera.position.set(.25, .25, -1);
            camera.lookAt(new THREE.Vector3(0, 0, 0));
            setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
        },
        // play transition
        function () {
            fadeMats(mats, [birdModel, flowerModel], 1, .75);
        },
        // play actions
        function () {
            gsapT1.clear();
            gsapT1_2.clear();

            //#region bird flap
            gsapT1_2.call(function () {
                switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[2]))
            })

            // filler
            gsapT1_2.to(sphereMesh.position, { duration: (Math.random() * 5 + 2), x:
sphereMesh.position }); //filler
```

```
                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
                })

                // filler
                gsapT1_2.to(sphereMesh.position, { duration: (Math.random() * 5 + 1), x:
sphereMesh.position }); //filler

                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
                })
                //#endregion

                //camera pan with rotObj parent
                /* let rotObj = new Object3D();
                scene.add(rotObj); */
                rotObj.add(camera);

                //let customEase = CustomEase.create("custom", "M0,0 C0,0 0.053,0.009 0.084,0.04
0.145,0.102 0.864,0.923 0.918,0.972 0.94,0.992 1,1 1,1 ");
                gsapT1.to(rotObj.rotation, { duration: 18, y: 6.28319, ease: 'linear', })

                // false = fade out | true = custom transition handler
                //gsapT1.call(function () { fadeOverride = true });
            }
            //#endregion
        ),
        new timelineObj(
            'bird is sand', 0,
            [birdModel],
            //#region anims
            // ready positions
            function () {
                //birdModel.position.set(0, 0, 0);
                rotObj.rotation.set(0, 0, 0);
```

```
    rotObj.position.set(0, 0, 0);

    camera.position.set(.25, .25, -1);
    camera.lookAt(new THREE.Vector3(0, 0, 0));
    setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
},
// play transition
function () {
    fadeMats(mats, gltfModels, 1, .75);
},
// play actions
function () {
    gsapT1.clear();
    gsapT1_2.clear();


    //#region bird flap
    gsapT1_2.call(function () {
        switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[2]))
    })

    gsapT1_2.to({}, { duration: (Math.random() * 5 + 2) }).call(function () {
        switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
    })

    gsapT1_2.to({}, { duration: (Math.random() * 5 + 1), }).call(function () {
        switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
    })
    //#endregion

    //#region POINTS
    let vertices = [];

    for (let i = 0; i < 5000; i++) {
        const x = THREE.MathUtils.randFloatSpread(2);
        const y = THREE.MathUtils.randFloatSpread(2);
```

```
        const z = THREE.MathUtils.randFloatSpread(2);

        vertices.push(x, y, z);
    }

    const material = new THREE.PointsMaterial({
        color: 0x888888,
        size: .0025,
        transparent: true,
        opacity: 1,
    });
    /* const points = new THREE.Points(buffGeometry, material); */


    let points = new THREE.Points(
        new THREE.BufferGeometry().setAttribute('position',
            new THREE.Float32BufferAttribute(vertices, 3)), material);

    scene.add(points);
    points.layers.set(0);

    /* class particleSystem {
                    constructor(_geometry, _uniforms, _vertices) {
                        this._geometry = geometry;
                        this._vertices = vertices;

                        this._material = new THREE.ShaderMaterial({
                            uniforms: _uniforms,
                            vertexShader:
document.getElementById(`vertexParticleShader`),
                            fragmentShader:
document.getElementById(`fragmentSimulation`),
                        });

                        this._points = new THREE.Points(_geometry, _material);
                        this.updateGeometry()
```

```
                        }

                        updateGeometry() {

                            this._geometry.setAttribute('position', new
THREE.Float32BufferAttribute(this._vertices, 3));
                            this._geometry.attributes.position.needsUpdate = true;
                        }

                        updateParticles() {

                        }
                    } */
                //#endregion

                //camera pan
                rotObj.add(camera);

                gsapT1.to(rotObj.rotation, {
                    duration: 12, ease: "linear", y: degToRad(360),
                })
                gsapT1.to(rotObj.position, {
                    duration: 8, ease: "linear", y: 1, onUpdate: function () {
                        camera.lookAt(new THREE.Vector3(0, 0, 0));
                    }
                }, `<`);
                gsapT1.to(material, { duration: 1, opacity: 1 }, '<');

                // move particles in random direction and update buffer??? see pevious versions

                // generate 5000 particles every frame
                gsapT1_2.to({}, {
                    duration: 10, ease: "power2.inOut",
                    onUpdate: function () {
                        scene.remove(points);
                        //fromVal.lerp(toVal, this.progress());
```

```
                vertices = [];

                for (let i = 0; i < 5000; i++) {
                    const x = THREE.MathUtils.randFloatSpread(2);
                    const y = THREE.MathUtils.randFloatSpread(2);
                    const z = THREE.MathUtils.randFloatSpread(2);

                    vertices.push(x, y, z);
                }

                points = new THREE.Points(
                    new THREE.BufferGeometry().setAttribute('position',
                        new THREE.Float32BufferAttribute(vertices, 3)), material);

                scene.add(points);
                points.layers.set(0);
                //console.log(this.progress());
            }
        }, 0);

        //sceneCompleted = true;
        //gsapT1.call(function () { fadeOverride = true });
    }
    //#endregion
),
new timelineObj('bird is water', 0,
    [birdModel, water],
    //#region anims
    // ready positions
    function () {
        camera.rotation.set()
        camera.position.set(.25, 2, -1);
        camera.lookAt(new THREE.Vector3(0, 0, 0));

        birdModel.rotation.set(0, 0, 0);
        setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
```

```
            },
            // play transition
            function () {
                fadeMats(mats, gltfModels, 1, 3);
            },
            function () {
                gsapT1.clear();
                gsapT1_2.clear();

                // disable all layers and set active to layer 0
                // TODO: toggle this part off for cool reflection|layer exploit
                scene.traverse(child => {
                    if (child instanceof THREE.Mesh) {
                        child.layers.disableAll();
                    }
                });

                let obj = { 'actors': [water, birdModel] };
                assignLayers(obj, 0);

                gsapT1.to({}, { duration: 1, });

                //#region bird and bird2 flap
                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[2]))
                })

                // filler
                gsapT1_2.to(sphereMesh.position, { duration: (Math.random() * 5 + 2), x:
sphereMesh.position }); //filler

                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
                })

                // filler
```

```
                gsapT1_2.to(sphereMesh.position, { duration: (Math.random() * 5 + 1), x:
sphereMesh.position }); //filler

                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
                })
                //#endregion



                // TEST_slowly move bird1 below water
                /* gsapT1.to(birdModel.position, { duration: 5, y: -.5 });
                gsapT1.to(birdModel.position, { duration: 5, y: 1.25 }); */

                //T1 to animate scale of water
                /* let valFrom = 0;
                let valTo = 1;
                gsapT1.to({}, {
                    duration: 5, ease: "power2.inOut",
                    onUpdate: function () {
                        water.material.uniforms['config'].value.w = MathUtils.lerp(valFrom,
valTo, this.progress());
                        //console.log(`water scale ` + this.progress());
                    }
                }, '<'); */

                //gsapT1.call(function () { fadeOverride = true });
            }
            //#endregion
        ),
        new timelineObj(
            'bird2 catches fish', 0,
            [birdModel, water],
            //#region anims
            // ready positions
            function () {
                // fade out fish and set position
```

```
                mats[mats.length - 1].opacity = 0;
                fishModel.position.set(0, .25, -1);

                birdModel.position.set(0, 0, 0);
                //birdModel.rotation.set(new THREE.Vector3(degToRad(180), degToRad(180), 0));

                bird2Model.position.set(1, 3, -25);

                // fade bird1 to .5
                birdModel.traverse(child => {
                    if (child.material) {
                        child.material.transparent = true;
                        child.material.opacity = .5;
                        console.log(child.material.opacity);
                    };
                });

                setDOFDistance(new THREE.Vector3(0, 0, 0), .25);
            },
            // play transition
            function () {
                fadeMats(null, [bird2Model], 1, .75);
            },
            // play actions
            function () {
                gsapT1.clear();
                gsapT1_2.clear();

                gsapT1.to(birdModel.rotation, { duration: .1, x: degToRad(180), y:
degToRad(180) }, '<');

                // disable all layers and set active to layer 0
                // TODO: toggle this part off for cool reflection|layer exploit
                scene.traverse(child => {
                    if (child instanceof THREE.Mesh) {
                        child.layers.disableAll();
```

```
                    }
                });

                // enable actors to layer
                let obj = { 'actors': [water, birdModel, bird2Model, fishModel] };
                gsapT1.call(assignLayers(obj, 0));

                //init pos
                gsapT1.to(camera.position, {
                    duration: 2, ease: "power2.inOut", x: 1, y: 2, z: -1.5,
                    onUpdate: function () {
                        camera.lookAt(new THREE.Vector3(0, .25, 0));
                    }
                }, `<`);

                //#region bird flap
                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[2]))
                })

                // filler
                gsapT1_2.to(sphereMesh.position, { duration: (Math.random() * 5 + 2), x:
sphereMesh.position }); //filler

                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
                })

                // filler
                gsapT1_2.to(sphereMesh.position, { duration: (Math.random() * 5 + 1), x:
sphereMesh.position }); //filler

                gsapT1_2.call(function () {
                    switchGLTFAnims(birdModel, birdMixer.clipAction(birdAnimations[3]))
                })
                //#endregion
```

```
                //T1 to animate scale of water
                /* let valFrom = 0;
                let valTo = .75;
                gsapT1.call(function () {
                    gsapT3.clear();
                    gsapT3.to({}, {
                        duration: 8, ease: "power2.inOut",
                        onUpdate: function () {
                            water.material.uniforms['config'].value.w = MathUtils.lerp(valFrom,
valTo, this.progress());
                            //console.log(`water scale ` + this.progress());
                        }, onComplete: function () {
                            console.log(`completed water scale`)
                        },
                    })
                }); */

                //#region start/stop fish anim
                let fishAnimLength = fishModel.activeClip._clip.duration;

                // add labels for fishEnter and fishExit
                gsapT1.addLabel(`fishEnter`, `+=0`)
                gsapT1.addLabel(`fishExit`, `+=${fishAnimLength}`)
                console.log(gsapT1.labels.fishEnter);
                console.log(gsapT1.labels.fishExit);

                // reset and play fish animation once faded in
                gsapT1.to({}, {
                    duration: 0,
                    onComplete: function () {
                        console.log(`unpause fishAnim`)
                    },
                }, `fishEnter`).call(function () {
                    fishModel.activeClip.reset();
                    fishModel.activeClip.paused = false;
```

```
                    fishModel.activeClip.play();
                });

                // fade in fish
                gsapT1.to(mats[mats.length - 1], {
                    duration: .5, opacity: 1, onComplete: function () {
                        console.log(`fade ${mats[mats.length - 1].name}`);
                    }
                }, `fishEnter`);

                // filler with duration, to pause fishAnim
                gsapT1.to({}, {
                    duration: .25,
                    onComplete: function () { console.log(`completed filler 2, pause fish
anim`) },
                }, `fishExit-=.25`).call(function () {
                    fishModel.activeClip.paused = true;
                });
                //#endregion

                // move to fish loc
                gsapT1.to(bird2Model.position, {
                    duration: fishAnimLength, ease: `linear`,
                    x: .45, y: 0.85, z: 1,
                }, `fishEnter`);

                //gsapT1.call(function () { fadeOverride = true });

                // fly away with fish
                let birdFishPos = [0, 2, 5];
                let birdFishDur = 2;
                gsapT1.to(bird2Model.position, {
                    duration: birdFishDur, ease: `power1.out`,
                    x: birdFishPos[0], y: birdFishPos[1], z: birdFishPos[2],
                }, `fishExit`);
                gsapT1.to(fishModel.position, {
```

```
                duration: birdFishDur, ease: `power1.out`,
                x: birdFishPos[0] - .44, y: birdFishPos[1] - .535, z: birdFishPos[2] - 2.1,
            }, `fishExit`);
        }
        //#endregion
    ),
    new timelineObj(
        'pan to watch bird2', 0,
        [water, birdModel, bird2Model, fishModel],
        //#region anims
        // ready positions
        function () {
            gsapT1.clear();
            camera.position.set(1, 2, -1.5);
            camera.lookAt(new THREE.Vector3(0, .25, 0));

            fishModel.activeClip.paused = true;
            fishMixer.setTime(0);

            // init bird2 Pos and fish pos at mixerTime = 0
            let birdFishPos = [0, 1.75, 2];

            bird2Model.position.set(birdFishPos[0], birdFishPos[1], birdFishPos[2]);
            fishModel.position.set(birdFishPos[0] + .25, birdFishPos[1] - .05, birdFishPos[2]
- .25)
            setDOFDistance(new THREE.Vector3(0, 1.75, 2), 1);
        },
        // play transition
        function () {
            fadeMats(mats, gltfModels, 1, .75);
        },
        // play actions
        function () {
            gsapT1.clear();
            //gsapT1_2.clear();
```

```
// disable all layers and set active to layer 0
// TODO: toggle this part off for cool reflection|layer exploit
scene.traverse(child => {
    if (child instanceof THREE.Mesh) {
        child.layers.disableAll();
    }
});

// enable actors to layer
let obj = { 'actors': [water, birdModel, bird2Model, fishModel] };
gsapT1.call(assignLayers(obj, 0));

//init camera pos
let vecFrom = camera.getWorldDirection(new THREE.Vector3());
let vecTo = new THREE.Vector3(0, 2, 10);
gsapT1.to({}, {
    duration: 8, ease: "power2.Out",
    onUpdate: function () {
        let state = vecFrom.lerp(vecTo, this.progress());
        camera.lookAt(state);
    },
});

// init birdPos 2
/* let birdFishPos = [0, 1.75, 2];
let birdFishDur = .1;
gsapT1.to(bird2Model.position, {
    duration: birdFishDur, ease: `power1.out`,
    x: birdFishPos[0], y: birdFishPos[1], z: birdFishPos[2],
}, `<`);
gsapT1.to(fishModel.position, {
    duration: birdFishDur, ease: `power1.out`,
    x: birdFishPos[0] - .44, y: birdFishPos[1] - .535, z: birdFishPos[2] - 2.1,
}, `<`); */

//#region bird2 flap
```

```
        gsapT1_2.call(function () {
            switchGLTFAnims(bird2Model, bird2Mixer.clipAction(bird2Animations[2]))
        })

        // filler
        gsapT1_2.to({}, {
            duration: (Math.random() * 5 + 2),
        }).call(function () {
            switchGLTFAnims(bird2Model, bird2Mixer.clipAction(bird2Animations[3]))
        })

        // filler
        gsapT1_2.to({}, {
            duration: (Math.random() * 5 + 1),
        }).call(function () {
            switchGLTFAnims(bird2Model, bird2Mixer.clipAction(bird2Animations[3]))
        })
        //#endregion
    }
    //#endregion
),
new timelineObj(
    'trees grow in wake of bird', 0,
    [bird2Model, fishModel, treeModel],
    //#region anims
    // ready positions
    function () {
        camera.position.set(1, 2, -1.5);
        camera.lookAt(new THREE.Vector3(0, .25, 0));
        setDOFDistance(new THREE.Vector3(0, .25, 0), 0)

        treeMixer.setTime(0);
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, 1.25);
```

```
},
// play actions
function () {
    gsapT1.clear();
    gsapT1_2.clear();
    camera.removeFromParent();

    //init pos
    gsapT1.to(camera.position, {
        duration: 2, ease: "power2.inOut", x: 7, y: 0, z: 0,
        onUpdate: function () {
            camera.lookAt(new THREE.Vector3(0, 0, 0));
            setDOFDistance(new THREE.Vector3(0, 0, 0), 0);
        }
    }, `<`);

    fishMixer.paused = true;
    fishMixer.setTime(0);


    //fishModel.rotation.x = radToDeg(90);
    // init birdPos 2
    let birdFishPos = [0, .5, -13];
    let birdFishDur = .1;
    gsapT1.to(bird2Model.position, {
        duration: birdFishDur, ease: `power1.out`,
        x: birdFishPos[0], y: birdFishPos[1], z: birdFishPos[2],
    }, `<`);
    gsapT1.to(fishModel.position, {
        duration: birdFishDur, ease: `power1.out`,
        x: birdFishPos[0] + .25, y: birdFishPos[1] - .05, z: birdFishPos[2] - .25,
    }, `<`);

    //#region bird2 flap
    gsapT1_2.call(function () {
        switchGLTFAnims(bird2Model, bird2Mixer.clipAction(bird2Animations[2]))
```

```
})

// filler
gsapT1_2.to({}, {
    duration: (Math.random() * 5 + 2),
}).call(function () {
    switchGLTFAnims(bird2Model, bird2Mixer.clipAction(bird2Animations[3]))
})

// filler
gsapT1_2.to({}, {
    duration: (Math.random() * 5 + 1),
}).call(function () {
    switchGLTFAnims(bird2Model, bird2Mixer.clipAction(bird2Animations[3]))
})
//#endregion

gsapT1.call(function () {
    treeMixer.setTime(0);
    treeModel.animations.forEach(animation => {
        var action = treeMixer.clipAction(animation);
        action.reset();
        action.loop = THREE.LoopOnce;
        action.clampWhenFinished = true;
        action.clamp
        action.play();
    });
});

birdFishPos = [0, .5, 15];
birdFishDur = 10;
gsapT1.to(bird2Model.position, {
    duration: birdFishDur, ease: `none`,
    x: birdFishPos[0], y: birdFishPos[1], z: birdFishPos[2],
}, `<`);
gsapT1.to(fishModel.position, {
```

```
            duration: birdFishDur, ease: `none`,
            x: birdFishPos[0] + .25, y: birdFishPos[1] - .05, z: birdFishPos[2] - .25,
        }, `<`);
    }
    //#endregion
),
new timelineObj(
    'boy fades back in', 0,
    [boyModel, windWispModel],
    //#region anims
    // ready positions
    function () {
        //.position.set(0, 0, 0);

        camera.position.set(8, 0, 0);
        camera.lookAt(new THREE.Vector3(0, 0, 0));
        setDOFDistance(new THREE.Vector3(0, 0, 0), 0)
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, .75);
    },
    // play actions
    function () {
        gsapT1.clear();
        gsapT1_2.clear();

        //init pos
        gsapT1.to(camera.position, {
            duration: 2, ease: "power2.inOut", x: 0, y: .25, z: 1,
            onUpdate: function () {
                camera.lookAt(new THREE.Vector3(0, .35, 0));
                setDOFDistance(new THREE.Vector3(0, .35, 0), 0);
            }
        }, `<`);
```

```
        // set position of boy
        // play hair animation
        gsapT1.to(boyModel.position, {
            duration: .1, x: 0, y: 0, z: 0,
        }, `<`).call(function () {
            switchGLTFAnims(boyModel, sitting_NLA);
            boyMixer.clipAction(boyAnimations[5]).play();
        })

        //gsapT1.call(function () { /* fadeOverride = true; */ });
    }
    //#endregion
),
new timelineObj(
    'camera stays on boy', 0,
    [boyModel, windWispModel],
    //#region anims
    // ready positions
    function () {
        boyModel.position.set(0, 0, 0);
        switchGLTFAnims(boyModel, sitting_NLA);
        boyMixer.clipAction(boyAnimations[5]).play();

        camera.position.set(0, .25, 1);
        camera.lookAt(new THREE.Vector3(0, .35, 0));
        setDOFDistance(new THREE.Vector3(0, .25, 0), 0)
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, .75);
    },
    // play actions
    function () {
        gsapT1.clear();
        gsapT1_2.clear();
```

```
        gsapT1.to({}, { duration: 1, });
    }
    //#endregion
),
new timelineObj(
    'a sandstorm kicks up', 0,
    [boyModel, windWispModel],
    //#region anims
    // ready positions
    function () {
        boyModel.position.set(0, 0, 0);
        switchGLTFAnims(boyModel, sitting_NLA);
        boyMixer.clipAction(boyAnimations[5]).play();

        camera.position.set(0, .25, 1);
        camera.lookAt(new THREE.Vector3(0, .35, 0));

        bloomPass.intensity = 2;
        bloomPass.luminanceThreshold = .5;
        bloomPass.blurPass.scale = 1;
        bloomPass.blurPass.width = window.innerWidth;
        bloomPass.blurPass.height = window.innerHeight;
    },
    // play transition
    function () {
        fadeMats(mats, gltfModels, 1, .75);
    },
    // play actions
    function () {
        gsapT1.clear();
        gsapT1_2.clear();

        //#region POINTS
        let vertices = [];
```

```
for (let i = 0; i < 5000; i++) {
    const x = THREE.MathUtils.randFloatSpread(2);
    const y = THREE.MathUtils.randFloatSpread(2);
    const z = THREE.MathUtils.randFloatSpread(2);

    vertices.push(x, y, z);
}

/* const buffGeometry = new THREE.BufferGeometry();
buffGeometry.setAttribute('position', new THREE.Float32BufferAttribute(vertices,
3));
buffGeometry.attributes.position.needsUpdate = true; */

const material = new THREE.PointsMaterial({
    color: 0x888888,
    size: .0025,
    transparent: true,
    opacity: 0,
});
/* const points = new THREE.Points(buffGeometry, material); */


let points = new THREE.Points(
    new THREE.BufferGeometry().setAttribute('position',
        new THREE.Float32BufferAttribute(vertices, 3)), material);

scene.add(points);
points.layers.set(0);


/* class particleSystem {
    constructor(_geometry, _uniforms, _vertices) {
        this._geometry = geometry;
        this._vertices = vertices;

        this._material = new THREE.ShaderMaterial({
```

```
                        uniforms: _uniforms,
                        vertexShader: document.getElementById(`vertexParticleShader`),
                        fragmentShader: document.getElementById(`fragmentSimulation`),
                    });

                    this._points = new THREE.Points(_geometry, _material);
                    this.updateGeometry()
                }

                updateGeometry() {

                    this._geometry.setAttribute('position', new
THREE.Float32BufferAttribute(this._vertices, 3));
                    this._geometry.attributes.position.needsUpdate = true;
                }

                updateParticles() {

                }
            } */
            //#endregion

            gsapT1.to(material, { duration: 1, opacity: 1 }, '<');

            // to move particles, finish the particleSystem class

            // generate 5000 particles every frame
            gsapT1_2.to({}, {
                duration: 2, ease: "power2.inOut",
                onUpdate: function () {
                    scene.remove(points);
                    //fromVal.lerp(toVal, this.progress());
                    vertices = [];

                    for (let i = 0; i < 5000; i++) {
                        const x = THREE.MathUtils.randFloatSpread(2);
```

```
                    const y = THREE.MathUtils.randFloatSpread(2);
                    const z = THREE.MathUtils.randFloatSpread(2);

                    vertices.push(x, y, z);
                }

                points = new THREE.Points(
                    new THREE.BufferGeometry().setAttribute('position',
                        new THREE.Float32BufferAttribute(vertices, 3)), material);

                scene.add(points);
                points.layers.set(0);
                //console.log(this.progress());
            }
        }, `<`);
    }
    //#endregion
),
new timelineObj(
    'zoom out, bloom intensifies', 0,
    [boyModel, windWispModel],
    //#region anims
    // ready positions
    function () {
        boyModel.position.set(0, 0, 0);
        switchGLTFAnims(boyModel, sitting_NLA);
        boyMixer.clipAction(boyAnimations[5]).play();

        camera.position.set(0, .25, 1);
        camera.lookAt(new THREE.Vector3(0, .35, 0));

        bloomPass.intensity = 2;
        bloomPass.luminanceThreshold = .5;
        bloomPass.blurPass.scale = 1;
        bloomPass.blurPass.width = window.innerWidth;
        bloomPass.blurPass.height = window.innerHeight;
```

```
            },
            // play transition
            function () {
                fadeMats(mats, gltfModels, 1, .75);
            },
            // play actions
            function () {
                gsapT1.clear();

                let fadeDur = 12;

                gsapT1.to(camera.position, {
                    duration: fadeDur, ease: "power2.inOut", x: 0, y: 0, z: 1.5,
                    onUpdate: function () {
                        camera.lookAt(new THREE.Vector3(0, .35, 0));
                    }
                }, `<`)

                //bloom to fill screen
                gsapT1.to(bloomPass, {
                    duration: fadeDur, ease: "power2.inOut", intensity: 50,
                }, `<`);
                gsapT1.to(bloomPass.blurPass, {
                    duration: fadeDur, ease: "power2.inOut", scale: 15, width: 50, height: 1080,
                }, `<`);

                /* console.log(bloomPass);
                console.log(bloomPass.blurPass); */


                // default bloom settings
                let implodeDur = .1;
                gsapT1.to(bloomPass, {
                    duration: implodeDur, ease: "power2.in", intensity: 50, luminanceThreshold:
0,
```

```
                });
                gsapT1.to(bloomPass.blurPass, {
                    duration: implodeDur, ease: "power2.in", scale: 12, width: 1080, height:
1080,
                }, `<`);

                gsapT1.to({}, { duration: fadeDur }).call(function () {
                    scene.traverse(child => {
                        if (child.material) {
                            child.material.transparent = true;
                            //child.material.opacity = 0
                            gsapT1.to(child.material, { duration: implodeDur, opacity: 0 }, '<');
                        };
                    });
                    gsapT1.to({}, {
                        duration: implodeDur, onComplete: function () {
                            scene.background = new THREE.Color(0x000000);
                        }
                    }, '<');

                })
            }
            //#endregion
        ),
    );
    //#endregion

    initNarration();
    initLayers();
    playScene(timelineClips[activeSceneNum], activeSceneNum);

    //#region TL GUI
    //toggle the GSAP timeline
    let playing = true;
    gui.add({ button: playing }, "button").name("play/pause").onChange(function () {
        if (playing) {
```

```
            gsapT1.pause();
        } else {
            gsapT1.play();
        }
        playing = !playing;
    });

    // continue to next scene
    gui.add({
        nextScene: function () {
            advanceScene();
        }
    }, 'nextScene');
    gui.add({
        previousScene: function () {
            previousScene();
        }
    }, 'previousScene');
    gui.add({
        restart: function () {
            restartScene();
        }
    }, 'restart');
    //#endregion

    /* gsapT1.timeScale(2);
    gsapT2.timeScale(2);
    gsapT3.timeScale(2); */
}

//#region SCENE_NAVIGATION

function advanceScene() {
    cursor.style.display = 'none';
    //console.log(`%c set cursor to none`, `#fefefe`)
    //console.log(`timelineClips length ${timelineClips.length}`)
```

```
    if (activeSceneNum < timelineClips.length - 1) {
        activeSceneNum++;
        //handle <p> swap for narration
        swapNarration(allNarration[activeSceneNum].innerHTML);
        //handle fade out, layers, and cursor
        playScene(timelineClips[activeSceneNum], activeSceneNum)
    } else {
        activeSceneNum = 0;
        swapNarration(allNarration[activeSceneNum].innerHTML);
        playScene(timelineClips[activeSceneNum], activeSceneNum)
    }
}

function previousScene() {
    cursor.style.display = 'none';

    if (activeSceneNum > 0) {
        activeSceneNum--;
        //handle <p> swap for narration
        swapNarration(allNarration[activeSceneNum].innerHTML);
        //handle fade out, layers, and cursor
        playScene(timelineClips[activeSceneNum], activeSceneNum)
    } else {
        activeSceneNum = 0;
        swapNarration(allNarration[activeSceneNum].innerHTML);
        playScene(timelineClips[activeSceneNum], activeSceneNum)
    }
}

function restartScene() {
    cursor.style.display = 'none';
    activeSceneNum = 0;
    swapNarration(allNarration[activeSceneNum].innerHTML);
    playScene(timelineClips[activeSceneNum], activeSceneNum)
}
```

```
function goToScene(sceneNum) {
    cursor.style.display = 'none';
    activeSceneNum = sceneNum;
    swapNarration(allNarration[activeSceneNum].innerHTML);
    playScene(timelineClips[activeSceneNum], activeSceneNum)
}

function playScene(sceneObj, layerNum) {

    //console.log(DOFPass);
    //console.log(DOFPass.getTarget());

    // !
    autoRot = false;
    _actionsComplete = false;

    console.log(`%c active scene: ${layerNum} ${sceneObj.name}`, 'color: #1BA5D8');

    gsapT3.clear();
    gsapT1.clear();
    rotObj.clear();

    // set repeat and play animations
    gsapT1.repeat(sceneObj.repeat);

    // fade out mats
    if (!fadeOverride) { fadeMats(mats, gltfModels, 0, .75); };

    gsapT3.call(function () {
        sceneObj.readyPositions();

        // !
        /* cameraInitial.x = camera.rotation.x;
        cameraInitial.y = camera.rotation.y; */
        cameraInitial.x = camera.position.x;
        cameraInitial.y = camera.position.y;
```

```
        // !
        autoRot = true;
        //console.log(`READY POSITIONS`);
    });

    // assign layers
    gsapT3.call(function () {
        assignLayers(sceneObj, layerNum);
        //console.log(`ASSIGN LAYERS`);
    });

    // reset fade override and play the scene
    gsapT3.call(function () {
        fadeOverride = false;

        // fade in mats —— add to new playTransition() function?
        //fadeMats(mats, gltfModels, 1, .75);

        sceneObj.playTransition();
        //console.log(`PLAY TRANSITION`);
        sceneObj.playActions();
        //console.log(`PLAY ACTIONS`);
        // !
        gsapT1.call(function () {
            _actionsComplete = true;
        });
    });
}

function initLayers() {
    scene.traverse(child => {
        if (child instanceof THREE.Mesh) {
            child.layers.disableAll();
        }
    });
```

```
}

function assignLayers(sceneObj, layerNum) {
    // disable all actors from all layers
    layerNum = 0;
    scene.traverse(child => {
        if (child instanceof THREE.Mesh) {
            child.layers.disableAll();
        }
    });

    // set layer actors
    for (let i = 0; i < sceneObj.actors.length; i++) {
        //console.log(sceneObj.actors[i]);
        sceneObj.actors[i].traverse(function (child) {
            child.layers.set(layerNum);
        });

    }
    camera.layers.set(layerNum);
}

function fadeMats(materials, models, opacity, duration) {
    //console.log(`%c fade mats, o: ${opacity}, d: ${duration}`, `color: #B5B5B5`);
    if (materials) {
        if (Array.isArray(materials)) {
            materials.forEach(mat => {
                gsapT3.to(mat, { duration: duration, opacity: opacity }, '<');
            });
        } else {
            gsapT3.to(materials, { duration: duration, opacity: opacity }, '<');
        }
    }

    if (models) {
        models.forEach(model => {
```

```
            model.traverse(child => {
                if (child.material) {
                    child.material.transparent = true;
                    gsapT3.to(child.material, { duration: duration, opacity: opacity }, '<');
                };
            });
        });
    }
}

//#endregion



function degToRad(deg) {
    return (deg * (Math.PI / 180))
}
//#endregion

//#region NARRATOR

function swapNarration(newText) {
    //get timeline2, clear t2, fade out and then in narration over duration .5s

    _swapComplete = false;

    let narration = document.querySelector(".narrator p");
    //console.log(`${newText.includes('span') ? 'has' : 'does not have'} span`);
    gsapT2.clear();
    let spans = spliceString(newText, '<span>');
    if (!spans) {
        //if spans is not found
        gsapT2.to(narration, { duration: .5, opacity: 0 });
        gsapT2.call(function () { narration.innerHTML = newText });
        gsapT2.to(narration, { duration: .75, opacity: 1 });
        gsapT2.to(narration, { duration: (newText.length / 25) * 1 });
    } else {
```

```
        //if spans[] is returned
        spans.forEach(span => {
            gsapT2.to(narration, { duration: .5, opacity: 0 });
            gsapT2.call(function () { narration.innerHTML = span });
            gsapT2.to(narration, { duration: .75, opacity: 1 });
            /** call redundant timeline function for a forced wait time,
             * duration ( number of characters / 25 ) * 1.25
             */
            gsapT2.to(narration, { duration: (span.length / 25) * 1, opacity: 1 });
        });
    }

    // gsapT2 = completed
    gsapT2.call(function () {
        _swapComplete = true;
    })
}
function initNarration() {
    //get array of all narration, get narrator, swap allNarration[i] with narrator on timer
    allNarration = document.querySelectorAll(".allNarration p");
    swapNarration(allNarration[activeSceneNum].innerHTML);
}
function spliceString(str, substr) {
    let flag = false;
    let indexes = [];
    let spans = [];

    // get index of all spans
    for (let i = 0; i < str.length - substr.length + 1; i++) {
        if (str.substring(i, substr.length + i) == substr) {
            indexes.push(i);
            flag = true;
        }
    }
    //split string into substrings
    for (let i = 0; i < indexes.length; i++) {
```

```
        spans.push(str.slice(indexes[i], indexes[i + 1]));
    };
    //console.log(spans);

    return (flag ? spans : false);
}

//#endregion

//#region SCENE COMPLETE
// on complete of gsapT1
var actionsComplete = false;
var swapComplete = false

Object.defineProperty(this, '_actionsComplete', {
    get: function () { console.log(`get actions`); return actionsComplete; },
    set: function (v) {
        actionsComplete = v;
        console.log('Actions completed: ' + v);
        //console.log(sceneCompleted());

        if (_swapComplete && _actionsComplete) {
            sceneCompleted();
        }
    }
});

Object.defineProperty(this, '_swapComplete', {
    get: function () { console.log(`get swap`); return swapComplete; },
    set: function (w) {
        swapComplete = w;
        console.log('Swap completed: ' + w);
        //console.log(sceneCompleted());

        if (_swapComplete && _actionsComplete) {
            sceneCompleted();
```

```
        }
    }
});

function sceneCompleted() {
    // queue the cursor fade to t2 after t1 has completed
    console.log(`scene completed?`);
    cursor.style.display = 'block';
    console.log(`%c cursor to block`, `color: #fefefe`);
    if (canBegin) {
        if (autoplay._enabled) {
            advanceScene();
        }
    } /* else if (!canBegin) {
        startButton.click();
        autoplay.lastMouseTime = clock.getElapsedTime();
        autoplay.setEnabled(false);
    } */
}
//#endregion



//#region MOUSE
const mouseObj = new THREE.Vector2(0, 0);
mouseObj.directionX = 0;
mouseObj.directionY = 0;
const targetMod = new THREE.Vector2();
//const cameraInitial = new THREE.Vector2();
const cameraInitial = new THREE.Vector3();
let timeDirStartX, timeDirDeltaX;
let timeDirStartY, timeDirDeltaY;
let vModX = 0;
let vModY = 0;

document.addEventListener('mousemove', onMouseMove, false);
```

```
function onMouseMove(event) {

    if (autoplay._enabled) {
        autoplay.setEnabled(false);
        //console.log(`disabling autoplay`);

    }
    autoplay.lastMouseTime = clock.getElapsedTime();

    //#region Mouse X
    // -1 = left, 1 = right
    mouseObj.oldX = mouseObj.directionX;

    let oldLocX = mouseObj.x;
    let newLocX = (event.clientX/*  - (window.innerWidth / 2) */);
    mouseObj.directionX = (oldLocX > newLocX) ? -1 : (oldLocX < newLocX) ? 1 : 0;
    //console.log(mouseObj.directionX);

    // determine the time delta since the last mouse direction change
    if (mouseObj.oldX != mouseObj.directionX) {
        timeDirStartX = clock.elapsedTime;
        vModX = 0;
        //console.log(`reset vMod`)
    }

    timeDirDeltaX = clock.elapsedTime - timeDirStartX;
    // delta 0 - .25 | vMod 0 - 1
    if (timeDirDeltaX < .25) {
        vModX = timeDirDeltaX * 4;
    } else {
        vModX = 1;
    }
    //console.log(`delta: ${timeDirDeltaX}`);
    //console.log(`vmod: ${vModX}`);

    // get new quadrant coordinates of mouse
```

```
    mouseObj.x = (event.clientX/*  - (window.innerWidth / 2) */);

    /* targetMod.x = (1 - mouseObj.x) * 0.0001; */

    //console.log(`       `)
    //console.log(mouseObj);
    //console.log(`target: ${round(targetMod.x)}, ${round(targetMod.y)}`)
    //console.log(`cam: ${round(camera.rotation.y)}, ${round(camera.rotation.x)}, $
{round(camera.rotation.z)}`)

    //move mouse
    if (autoRot) {
        //camera.eulerOrder = 'YXZ';
        //camera.rotation.x = (targetMod.y) + cameraInitial.x;
        //camera.rotation.y = (targetMod.x) + cameraInitial.y;

        //camera.rotateY((targetMod.x) + cameraInitial.y);
        //camera.rotation.set(new THREE.Euler(0, 1, 1.57, 'XYZ');)

        // map range 0-1-0 = sin ( mouseX / (max/π))
        //let factor = 0.01 * (Math.sin(mouseObj.x / ((window.innerWidth / 2) / Math.PI)));
        let factor = mouseObj.directionX * Math.abs(
            0.005 * (Ma       th.sin(mouseObj.x / ((window.innerWidth) / Math.PI))));
        camera.translateX(factor * vModX);
        //camera.translateX(factor);

        //console.log(factor);
    }
    //#endregion

    //#region Mouse Y
    // -1 = left, 1 = right
    mouseObj.oldY = mouseObj.directionY;

    let oldLocY = mouseObj.y;
    let newLocY = (event.clientY/*  - (window.innerHeight / 2) */);
```

```
    mouseObj.directionY = (oldLocY > newLocY) ? 1 : (oldLocY < newLocY) ? -1 : 0;
    //console.log(mouseObj.directionY);

    // determine the time delta since the last mouse direction change
    if (mouseObj.oldY != mouseObj.directionY) {
        timeDirStartY = clock.elapsedTime;
        vModY = 0;
        //console.log(`reset vMod`)
    }

    timeDirDeltaY = clock.elapsedTime - timeDirStartY;
    // delta 0 - .25 | vMod 0 - 1
    if (timeDirDeltaY < .25) {
        vModY = timeDirDeltaY * 4;
    } else {
        vModY = 1;
    }
    //console.log(`delta: ${timeDirDeltaY}`);
    //console.log(`vmod: ${vModY}`);

    // get new quadrant coordinates of mouse
    mouseObj.y = (event.clientY/*  - (window.innerHeight / 2) */);

    targetMod.y = (1 - mouseObj.y) * 0.0001;

    //console.log(`       `)
    //console.log(mouseObj);
    //console.log(`target: ${round(targetMod.Y)}, ${round(targetMod.y)}`)
    //console.log(`cam: ${round(camera.rotation.y)}, ${round(camera.rotation.Y)}, $
{round(camera.rotation.z)}`)

    //move mouse
    if (autoRot) {
        let factor = mouseObj.directionY * Math.abs(
            0.0035 * (Math.sin(mouseObj.y / ((window.innerHeight) / Math.PI))));
        camera.translateY(factor * vModY);
```

```
        //camera.translateY(factor);

        //console.log(factor);
    }
    //#endregion

}

function round(num) {
    var m = Number((Math.abs(num) * 100).toPrecision(15));
    return Math.round(m) / 100 * Math.sign(num);
}

//#endregion




/**
 * Animate
 */
const tick = () => {

    //#region BASIC

    stats.begin()



    // Update Uniforms
    uniforms['time'].value = performance.now() / 1000;
    uniforms['resolution'].value = [window.innerWidth, window.innerHeight];

    // Update objects
    //wispModel.rotation.x = Math.sin(clock.getElapsedTime());

    // init GSAP only when models are loaded
    if (!mixerLoaded) {
```

```
    if (canBegin && boyMixer && sandWispModel && flowerModel && birdMixer) {
        initTimeline();
        mixerLoaded = true;
        console.log(`%c mixer and timeline loaded`, 'color: #B5B5B5');

    }
} else {
    // Update animation timing
    let mixerUpdateDelta = clock.getDelta();
    for (let i = 0; i < mixers.length; i++) {
        mixers[i].update(mixerUpdateDelta);
    }
    sandWispModel.rotation.y = clock.getElapsedTime();
}

//#endregion



// Update stats
stats.update();

// Update Orbital Controls

if (!isRotating) { controls.update(); }

// Render
effectComposer.render();
//renderer.render(scene, camera);

stats.end()

autoplay.tickAP();
//console.log(autoplay.i);

// Call tick again on the next frame
requestAnimationFrame(tick);
```

```
}



initObjects();
initScene();
tick();
```