Andrea Leganza
MAT. 788513

Final project

12/09/2022

1 Scene description

The scene is a diorama of an island placed in the middle of the ocean, the scene can be switched to day and night mode showing/hiding different elements.

2 Technologies

The project was developed using Three.js, HTML, CSS and JS

3 Libraries

The following libraries were used:

- Three.js
- Tween.js
- dat.gui

4 Assets

Props were dowloaded from sketchfab.org, cleaed and their topology modified in Blender and then exported in GLTF format:

- Castle `https://sketchfab.com/3d-models/sea-keep-lonely-watcher-09a15a0c14cb4accaf060a92bc70413d`
- Phoenix `https://sketchfab.com/3d-models/phoenix-king-d04bb96c5fb846c68b13754a8ece96a2`
- Boat `https://www.cadnav.com/3d-models/model-41627.html`
- Crab `https://sketchfab.com/3d-models/animated-crab-bbfe14b8698d4cdf901f90536b7d4e1e`
- Pricess `https://sketchfab.com/3d-models/low-poly-princess-rigged-9ab4bcd080b34583a1f837a06cd3efb0`

All models required to be cleaned up, their structure and original materials to be modified too. To improve performances the polygons of the models were reduced to different ratios.

All the animated models were rigged with digital bones most of the time using a single model mesh so in Blender these models were all separated as needed to accomplish the task to animate "manually" in three.js

Leaves on the water were created using a simple transparent plane and applying on realtime downloaded textures, while fireflies where created using a plane with random color.

5 User interface

The GUI was developed using DAT.GUI, the whole interaction is controlled by javascript events. It provides controls to manage mosto of the common interactions aspect of the scene:

- Camera: position and rotation
- Controls: speed parameters, autorotate parameters

- Day/night switch, fog parameters, shadowmap on/off
- Ambient light: on/off, intensity and color
- Torches: on/off, animation, color, intensity
- Spotlight: on/off, shadows, color, intensity, distance, angle, penumbra, position
- Sky: on/off
- Water: on/off
- Particles: night fireflies on/off
- Postprocessing: on/off bloom
- Audio: on/off, volume
- Debug Console Debug: on/off

(a) Menu **(b)** Menu **(c)** Menu

Figure 1: Menu views

6 Day/Night mode

Switching to day/night mode using the GUI button lets to switch to different look of the scene, showing/hiding different elements:

- Day: Phoenix flies around the castle, sky rotates around, leaves float, water tide
- Night: A boat travels around the castle, night sky rotates, fireflies fly around, torches lights up and animate their flame, a crab eat and moves, water tide
- The sky changes its texture from a cloudy day to a night sky.
- Fog changes to exp mode during night to increase the night look darkening the whole scene.
- Switching to night/day also customizes the lights with different colors and intensities to improve the scene look.

Figure 2: The DAY mode look using MODEL water shader

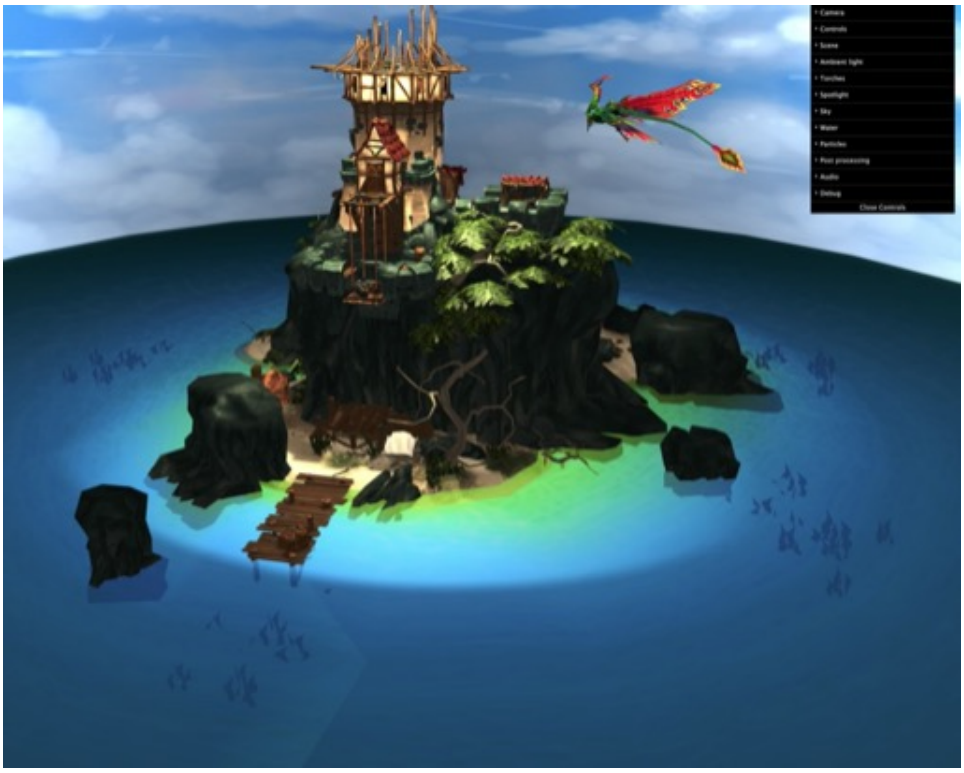


Figure 3: The NIGTH mode look using MODEL water shader

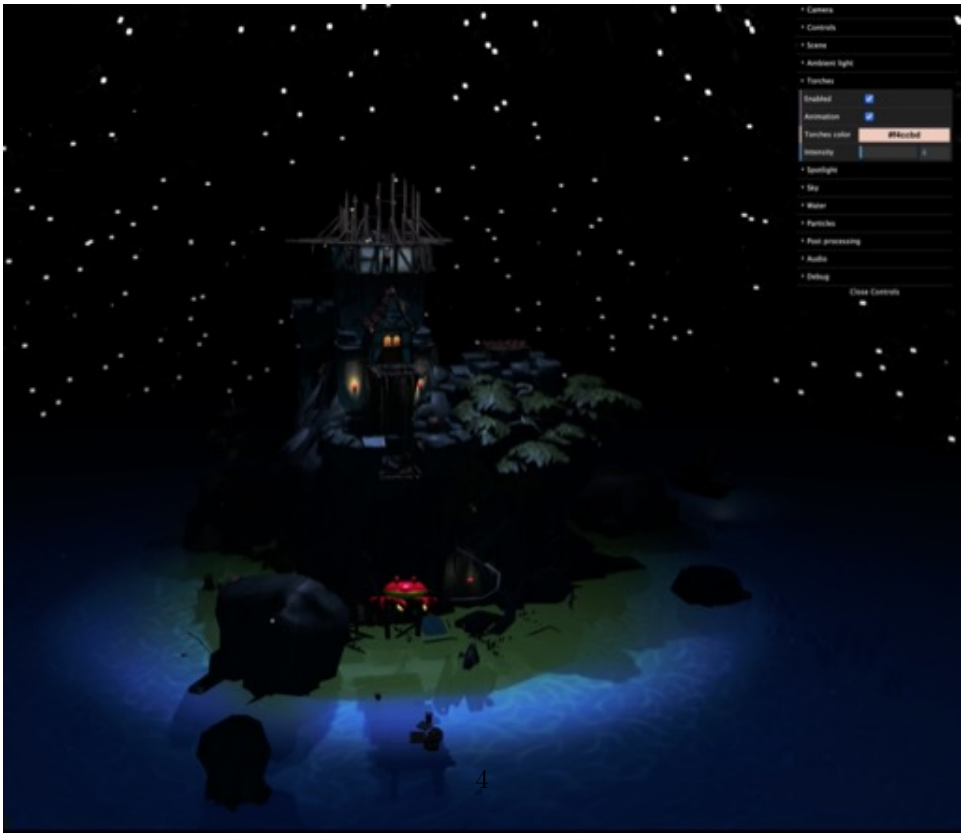
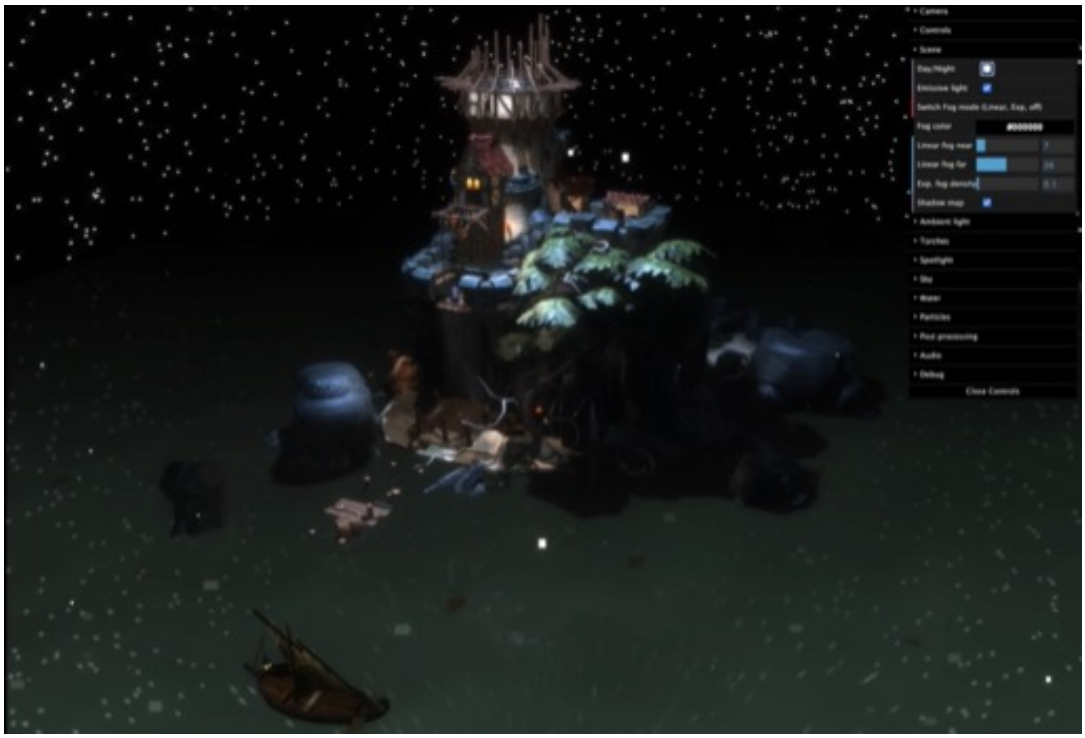


Figure 4: The DAY mode look using THREE.JS EXAMPLE water shader

Figure 5: The NIGTH mode look using THREE.JS EXAMPLE water shader



7 Sea

Figure 6: The model sea

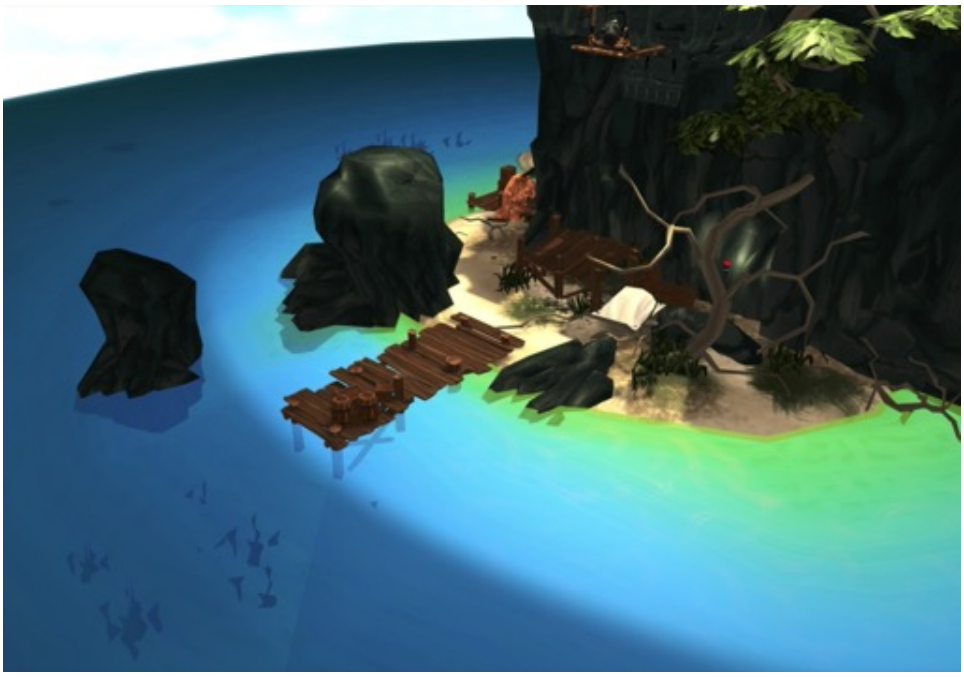


Figure 7: The THREE.JS/examples sea

Animation code:

```
function animateLiquid(target){

        if (liquidTween!=null){
            TWEEN.remove(liquidTween)
            liquidTween = null
        }

        if (debugMode){
            console.log("Animating water: "+target.name)
        }
        target.position.y = target == water ? -0.1 : 0;

        liquidTween = new TWEEN.Tween(target.position)
            .to({x: 0,
                 y : target == water ? 0 : 20,
                 z : 0},
            10000)
            // .delay (1000)
             .yoyo(true)
             .repeat(Infinity)
             .easing(TWEEN.Easing.Quadratic.InOut)
             /*.onUpdate(function () {

            })*/
             .start();
    }
```

8 Torches

Figure 8: One of the torches

Torches are detected during mesh loading and to them are added:

- A spotlight is added with related animation tween
- A sphere with some faces to mimic a Platonic solid to simulate the flame

To set the color of the flame a custom shader was made, which interpolates between two colors during the time (an uniform updated during the animation loop using a deltatime value) :

```
const sphere = new THREE.SphereGeometry( 3, 4, 2 );

          const mat = new THREE.ShaderMaterial({
              uniforms: {
                  u_time: { type: "f", value: 0 }
              },
              vertexShader: `
                  varying vec2 vUv;
                  uniform float u_time;

                  void main() {
```

```
            vUv = uv;

            mat4 scale = mat4(vec4(1.0+sin(u_time)*0.1,0.0,0.0,0.0),
                              vec4(0.0,1.0+sin(u_time)*0.3,0.0,0.0),
                              vec4(0.0,0.0,1.0+sin(u_time)*0.5,0.0),
                              vec4(0.0,0.0,0.0,1.0));
            gl_Position = projectionMatrix * modelViewMatrix * scale * \
            vec4(position,1.0);
        }
    `,
    fragmentShader: `
     uniform float u_time;
     varying vec2 vUv;
     void main() {
      gl_FragColor = vec4(mix(vec3(1,0,0), vec3(1,0.65,0), vUv.y+ \
      sin(u_time)), 1.0);
    }
`
});
```

9 Leaves

Leaves were created with a simple double side transparent plane usig a custom vertex/fragment shader, loading a random local stored leaf texture; they float around the scene with a simple Math.sin movement which starts from a random generated value and the position is updated via the shader position passing the total amount of time elapsed, this approach moves the calculations to the GPU improving performances (istead of updating the mesh.position in the animate loop).

Figure 9: The leaves on the surface

```
const geometry = new THREE.PlaneGeometry( 1, 1 );

   const textureName = 'Textures/leaf'+(Math.floor(Math.random() * 3) + 0)+'.png';

   const texture = new THREE.TextureLoader().load(textureName) ;

   const shaderMat = new THREE.ShaderMaterial({
           uniforms: {
               u_time: { type: "f", value: 0 },
               delta: { type: "f", value: 0 },
               colorTexture: { type: 't', value:texture}
           },
           vertexShader: `
               varying vec2 vUv;
               uniform float u_time;
               uniform float delta;
               uniform sampler2D colorTexture;

               void main() {
                   vUv = uv;

                   vec4 newPositon = vec4(sin(delta + u_time)*0.1,0.0,0.0,0.0);

                   gl_Position = (projectionMatrix * modelViewMatrix * \
                   (position,1.0)) + newPositon;
               }
           `,
           fragmentShader: `
               varying vec2 vUv;
               uniform sampler2D colorTexture;

               void main() {
                   vec4 color = texture2D( colorTexture, vUv );
                   gl_FragColor = color * vec4(0.3,0.3,0.3,1.0);
               }
       `
       });
       shaderMat.side = THREE.DoubleSide
       shaderMat.transparent = true
       shaderMat.depthTest =  true
       shaderMat.depthWrite = false

   var leavesCoordPosition = [];

   for (var i= 0; i <30; i++){
       var ranXNum = Math.ceil(Math.random() * 7) * (Math.round(Math.random()) ? 1 : -1)
       var ranYNum = Math.ceil(Math.random() * 7) * (Math.round(Math.random()) ? 1 : -1)

       const pos = new THREE.Vector3(ranXNum,0.01,ranYNum)
       leavesCoordPosition.push(pos);
```

```
}

leavesCoordPosition.forEach(position => {

    if (debugMode){
        console.log(position);
    }

    const mesh = new THREE.Mesh( geometry, shaderMat );

    mesh.rotation.z =  Math.random() * Math.PI * 2;

    mesh.rotation.x = target == water ? 0 : Math.PI/2;

    mesh.position.set(  position.x*(target == water ? 1 : 100),
                        position.y,
                        position.z*(target == water ? 1 : 100));

    let s = Math.random() *(target == water ? 0.1 : 30) + 0.1;
    mesh.scale.set(s, s, s);
    mesh.userData.delta = Math.random() * Math.PI * 2;
    mesh.material.uniforms["delta"].value = mesh.userData.delta

    floatingObjects.push(mesh);

    target.add(mesh);
});
```

10 Fireflies

Fireflies were developed as simple squares of different sizes and colors embbeded inside a BufferGeometry with a PointsMaterial, thei change color during the render loop.

Figure 10: Fireflies flying around the night scene

```
function placeFireFlies(){
  particleMode = ParticleMode.on

        if (particlestGroup != null){

            particlestGroup.visible = true

            return
        }

        particlestGroup = new THREE.Group();

        const geometry = new THREE.BufferGeometry();
        const vertices = [];

        const textureLoader = new THREE.TextureLoader();

        const sprite = textureLoader.load( '' );

        for ( let i = 0; i < 100; i ++ ) {

            const x = Math.random() * 9 - 4.5;
            const y = Math.random() * 9 - 4.5;
            const z = Math.random() * 9 - 4.5;

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

        geometry.setAttribute( 'position', new THREE.Float32BufferAttribute( vertices, 3 ) );
```

```
particleParameters = [
    [[ 1.0, 1.0, 1.0 ], sprite, 0.05 ],
    [[ 0.86, 0.86, 0.67], sprite, 0.02],
    [[ 0.3, 0.3, 0.3], sprite, 0.03 ],
];

for ( let i = 0; i < particleParameters.length; i ++ ) {

    const color = particleParameters[ i ][ 0 ];
    const sprite = particleParameters[ i ][ 1 ];
    const size = particleParameters[ i ][ 2 ];

    particleMaterials[ i ] = new THREE.PointsMaterial( { size: size, blending:
     THREE.AdditiveBlending, depthTest: false, transparent: true } );
    particleMaterials[ i ].color.setHSL( color[ 0 ], color[ 1 ], color[ 2 ] );

    const particles = new THREE.Points( geometry, particleMaterials[ i ] );

    particlesEffects.push(particles);

    particlestGroup.add( particles );
}

scene.add(particlestGroup)
```

11 Animations

Animations were implemented with different techniques:

- Updating rotation/position properties of mesh directly in the render loop
- Updating UV offset
- Using Tween.JS
- Updating shader properties

The following elements were animated:

- Sky: rotates on z axis on day, on z and y axis during night (using uv offset)
- Water: translates up/down to simulate low/high tide with an easing
- Torches: flame scales and rotates, their light glows randomly
- Leaves on the water: oscillate on the surface and float vertically
- Phoenix (day mode) flies and rotate around a spline (CatmullRomCurve3)
- Boat (night mode) rotate around castle and floats up/down
- Boat lantern (night mode): oscillates during boat movement

- Crab (night mnode): every leg moves, claws move, crab moves left/right, crab lookA the camera
- Fireflies particles: randomly move and change colors

12 Phoenix

The original model was rigged and was a sigle mesh, the first operation that was done on it was to remove the whole rigging hierarchy and split the different segments in Blender, then a proper hierarchy was created starting from the body. To speed up the animation process "custom properties" were added to each of the mesh segments to pass axis, easing, delay and movement direction which were read in three.js by accessing the userdata property; these properties were then used by Tween.js to animate as needed. All the meshes required to set an ad hoc pivot to be properly aimated.

Figure 11: The phoenix structure

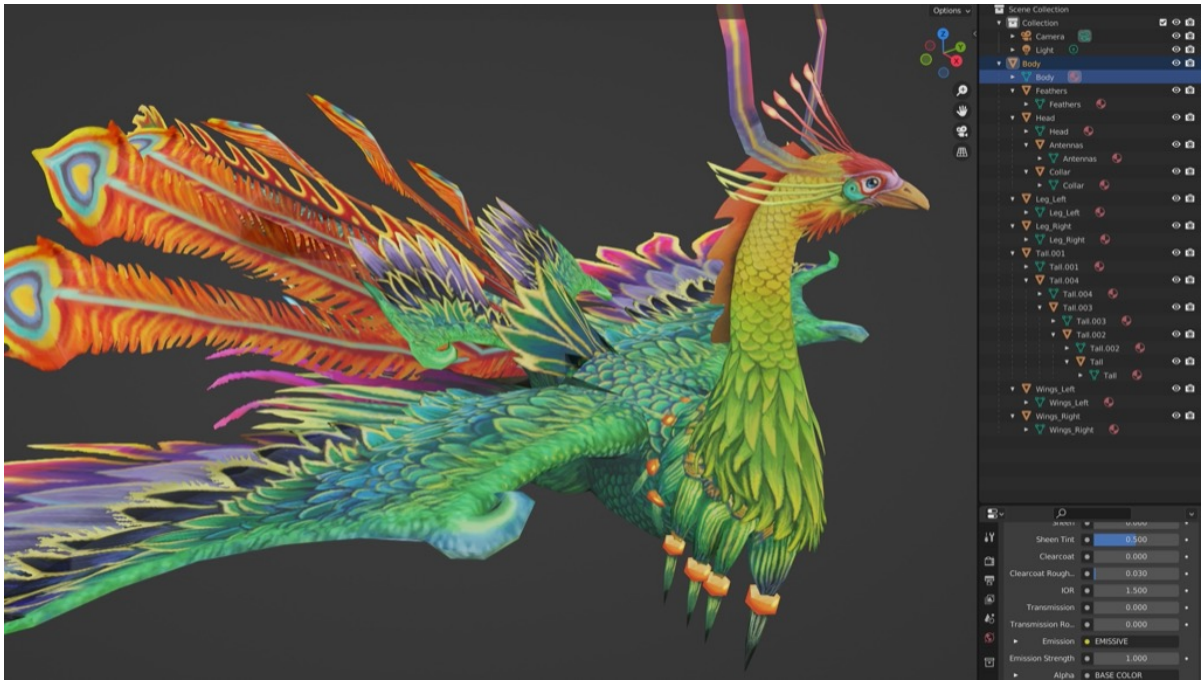
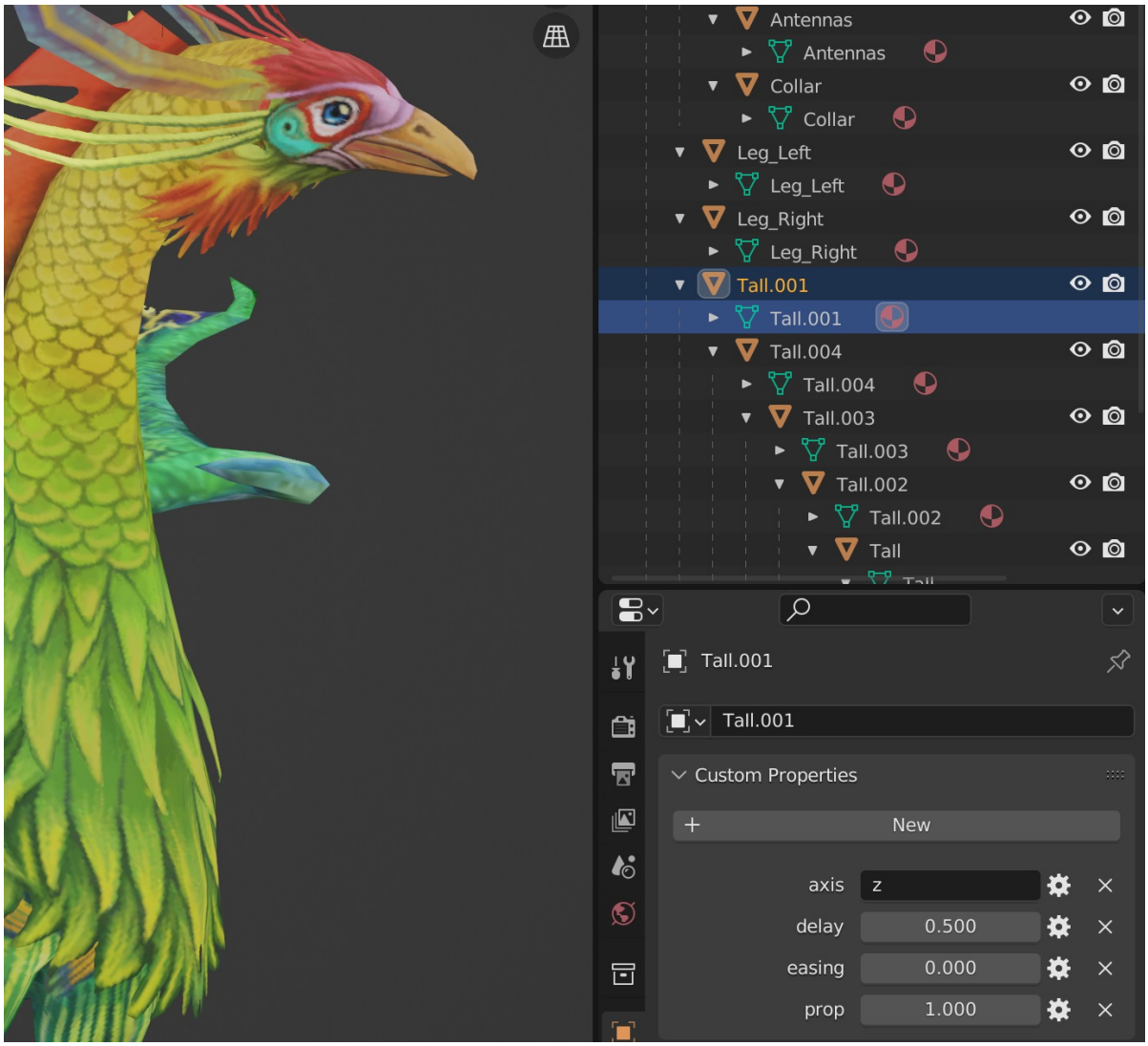


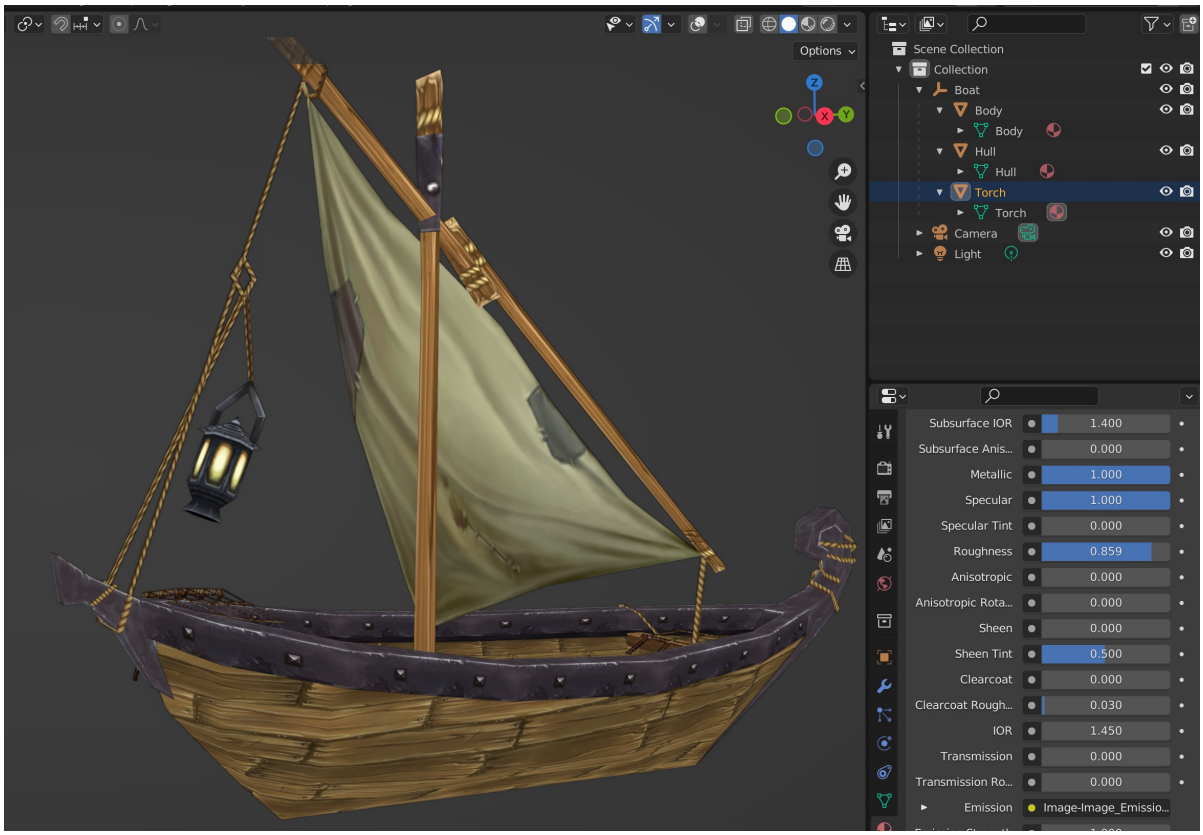
Figure 12: Custom properties used to pass data to Three.js



13 Boat

The boat was cleaned up, the pivot was changed to a proper position, the lantern was separated from the boat and the pivot set in the correct position for both of them to proper support the animation process.

Figure 13: The boat structure



14 Princess

The princess was cleaned up, the mesh was a serie of geometries, a new hierarchy was created, its poly count was reduced using Blender modifier; all the pivots were changed to match a proper position; crab legs are animated using Tween.js using the same approach of the Phoenix so using model passed custom properties. The princess updates it's rotation using the mesh.lookAt(camera.position) so it's always looking towards the user.

Figure 14: The princess structure

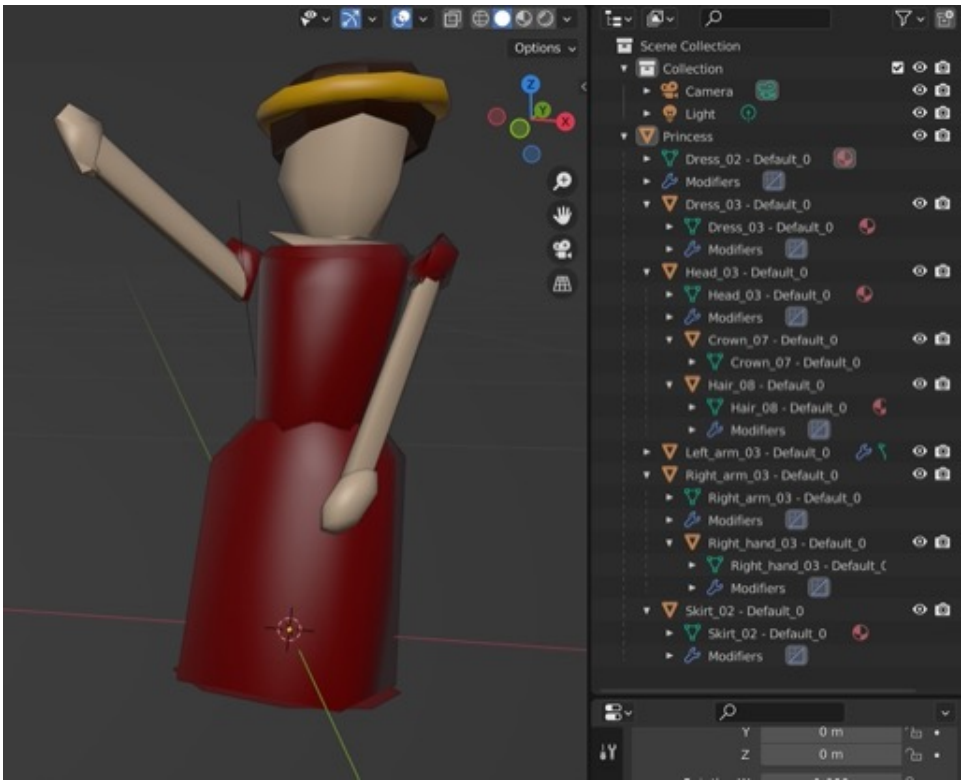
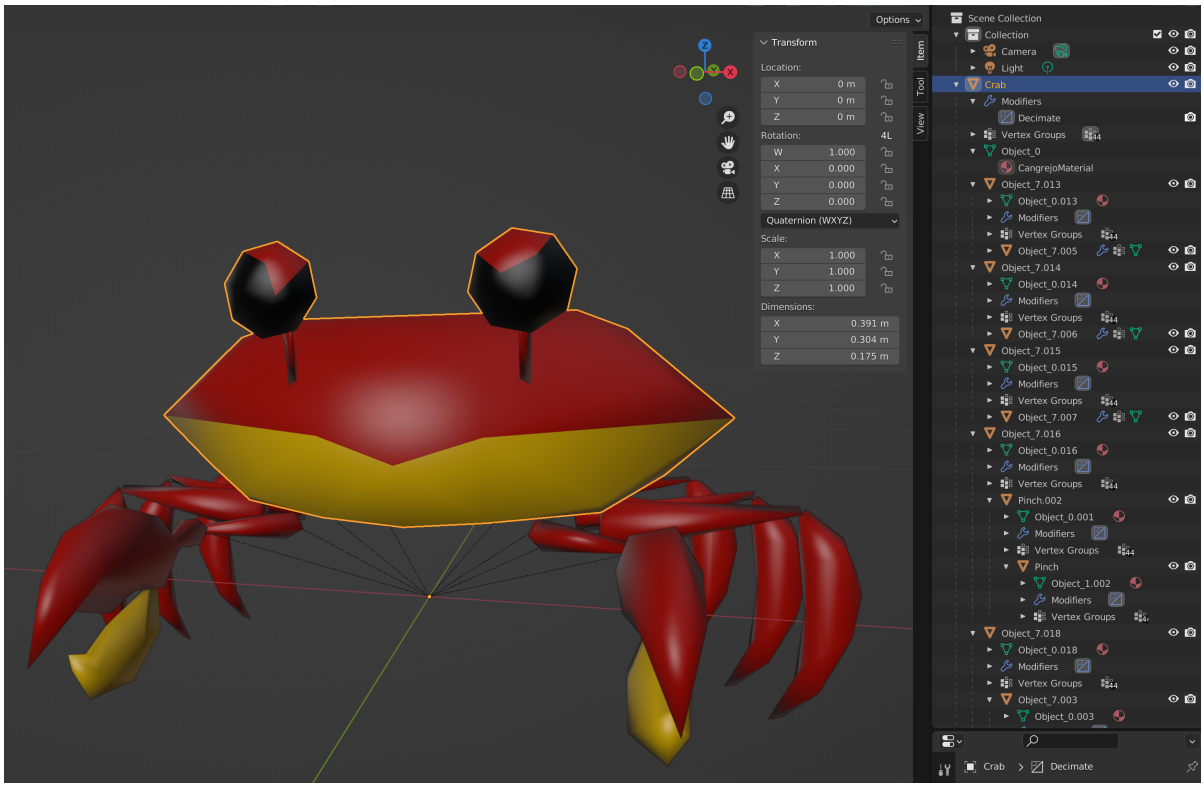


Figure 15: The princess waving

15 Crab

The crab was cleaned up, the mesh was an unique geometry, so it was splitted in segments with a new hierarchy and its poly count was reduced using Blender modifier; all the pivots were changed to match a proper position; crab legs are animated using Tween.js using the same approach of the Phoenix so using model passed custom properties. The crab updates it's rotation using the mesh.lookAt(camera.position) so it's always looking towards the user.

Figure 16: The crab structure




```
function animateCrabMesh(mesh, axis, delta, delay, easing, speed = 1){

if (debugMode){
    console.log("Animating crab: "+mesh.name);
}

const tween = new TWEEN.Tween(mesh.rotation)
    .to({[axis]:mesh.rotation[axis] + 0.2*delta},1000*speed)
    .yoyo(true)
    .delay(delay != null ? delay*1000 : 0)
    .repeat(Infinity)
    .easing(easing != null ? (easing == 1 ? TWEEN.Easing.Linear.None : TWEEN.Easing.Quadratic.InOut)
    .start()

    crabTweens.push(tween)
```

```
}
```

16 Textures

The following textures were used:

- Base/Albedo map
- Normal map
- Emissive map

For some models the normal map was generated from base/albedo map using an online service (`https://cpetry.github.io/NormalMap-Online/`); for emission some maps were generated by hand modifying in Krita the base map removing opaque areas; the emissive intensity effect was increased during the load of the models and a GUI option is available to change this value.

17 Lights

The lights used in the scene were:

- An Ambient Light to light the whole scene
- A Spotlight to light from the top the castle
- Four point lights for each torch lights to simulate the light of the torches
- A spot light on the boat

18 Post processing

A bloom effect is optionally available in the "post processing" menu area, different settings are available.

Figure 17: The bloom effect off

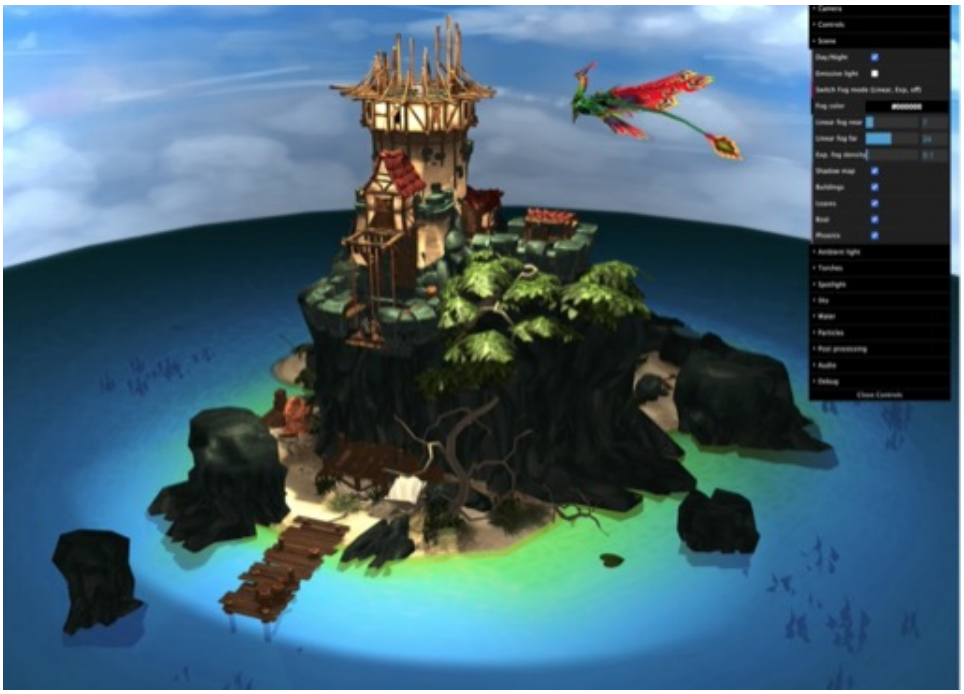
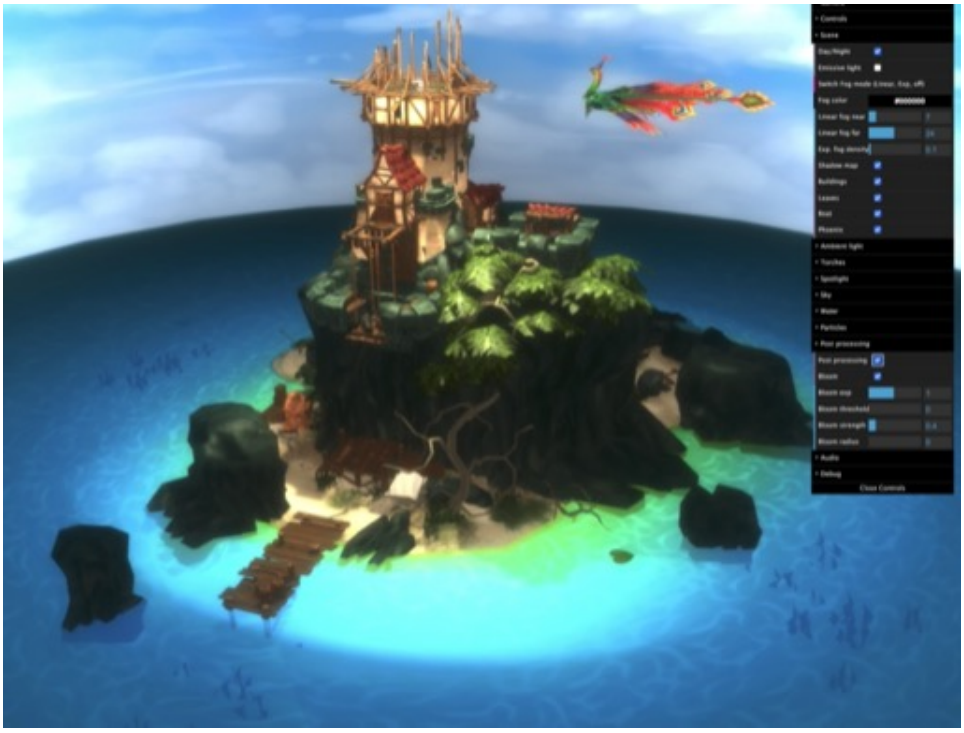


Figure 18: The bloom effect on



19 Audio

A seashore audio loop sound effect is provided, during the day mode a seagull sound is played too, while during the night a crickets soud is played.

20 Render loop

```
if (loading){
          return
      }

      deltaTime = clock.getDelta();

      totalTime+=deltaTime;

      if (skyEmbedded!=null && skyEmbedded.visible) {

          if(skyTime == SkyTime.day) {
              skyEmbedded.material.map.offset.x -= 0.001*deltaTime;
          }
          else {
              skyEmbedded.material.map.offset.x -= 0.001*deltaTime;
              skyEmbedded.material.map.offset.y += 0.001*deltaTime;
          }
      }

      if (waterUniforms!=null && water!=null) {
          waterUniforms[ 'time' ].value += 0.1*deltaTime;
      }

      if (boatGroup!=null) {
          if (boatGroup.parent == water){
              boatGroup.rotation.y += 0.02*deltaTime;
              boatGroup.position.z = Math.cos(totalTime)*0.05;
          }
          else if (boatGroup.parent==sea) {
           boatGroup.rotation.y += 0.02*deltaTime;
           boatGroup.position.z = Math.cos(totalTime)*0.05;
          }
      }

      //Fireflies
      if (particlestGroup != null && particlestGroup.visible){

          const time = Date.now() * 0.00001;

          for ( let i = 0; i < particlesEffects.length; i ++ ) {
              particlesEffects[i].rotation.y = time * ( i < 4 ? i + 1 : - ( i + 1 ) );
          }
```

```
    for ( let i = 0; i < particleMaterials.length; i ++ ) {
        const color = particleParameters[ i ][ 0 ];
        const h = ( 360 * ( color[ 0 ] + time ) % 360 ) / 360;
        particleMaterials[ i ].color.setHSL( h, color[ 1 ], color[ 2 ] );

    }
}

//Torchlights  animation
for (const torch of torchLights){
    if (torch.light.visible){
        torch.mesh.material.uniforms["u_time"].value += deltaTime;
    }
}

//Princess
if (princess!=null && princess.visible) {
    princess.lookAt(camera.position)
    princess.rotation.z = 0
    princess.rotation.x = 0
}


//Phoenix animation
if (phoenix!=null) {
    movePhoenixAroundPath(deltaTime*0.05);
}

//Leaves animation
floatingObjects.forEach(obj => {
    obj.material.uniforms["u_time"].value = totalTime
    //obj.position.y+= Math.sin(obj.userData.delta + totalTime) * 0.001;
});

//Crab
if (crab!=null && crab.visible) {
    crab.lookAt(camera.position)
    crab.rotation.z = 0
    crab.rotation.x = 0
}

requestAnimationFrame( animate );
!postProcessingMode ? renderer.render( scene, camera ) : \
 postProcessing.composer.render(deltaTime);
TWEEN.update();

stats.update()
```

21 Notes

The performances during camera look aren't great, performances should be improved: using lightmapping, storing shadows and reducing polygon count, also moving most of the animations to the shaders will improve the whole experience reducing render time. An alternative version of the water is provided enabling the proper option available under the "sea" menu voice (the sea shader is the one available from examples folder, vertical animation is made using tween.js).