Dawntime – A three.js implementation of crepuscular rays

Computer Graphics and 3D Course Project

Alberto Baldrati, Giovanni Berti

Supervisor: Prof. Stefano Berretti

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze

Index

Introduction

Introduction

- Crepuscular rays or *God rays* are sunbeams that originate when the sun is below the horizon, during twilight hours
- Crepuscular rays are noticeable when the contrast between light and dark is most obvious.
- Crepuscular rays are an instance of volumetric light scattering
- We implement a volumetric light scattering effect based on the postprocessing technique described by Kenny Mitchell.

Photo Example

Physical Description

Physical Description

Basic model

- To calculate the illumination at each pixel, we must account for scattering from the light source to that pixel:

$$L(s,\theta) = L_0\, e^{-\beta_{ex} s} + \frac{1}{\beta_{ex}} E_{sun}\, \beta_{sc}(\theta)\, (1 - e^{-\beta_{ex} s}) \tag{1}$$

- Where:
  - L_0 is the illumination of the light source
  - s is the distance traveled through the media
  - θ is the angle between the ray and the light source
  - E_{sun} is the irradiance of the light source
  - β_{ex} is the extinction constant composed of light absorption and out-scattering properties
  - β_{sc} is composed of light absorption and out-scattering properties

Physical Description

Accounting for scene geometry

- $D(\phi)$ represents the percentage light occluded by other objects in the scene. We can extend the previous equation:

$$L(s, \theta, \phi) = (1 - D(\phi))\, L(s, \theta) \tag{2}$$

- In screen space, we don't have full volumetric information to determine occlusion

- We estimate the probability of occlusion at each pixel by averaging samples along a ray to the light source in image space

$$L(s, \theta, \phi) = \sum_{i=0}^{n} \frac{L(s_i, \theta_i)}{n} \tag{3}$$

Physical Description
Shader approximation

- We can parametrize the previous summation in the following way:

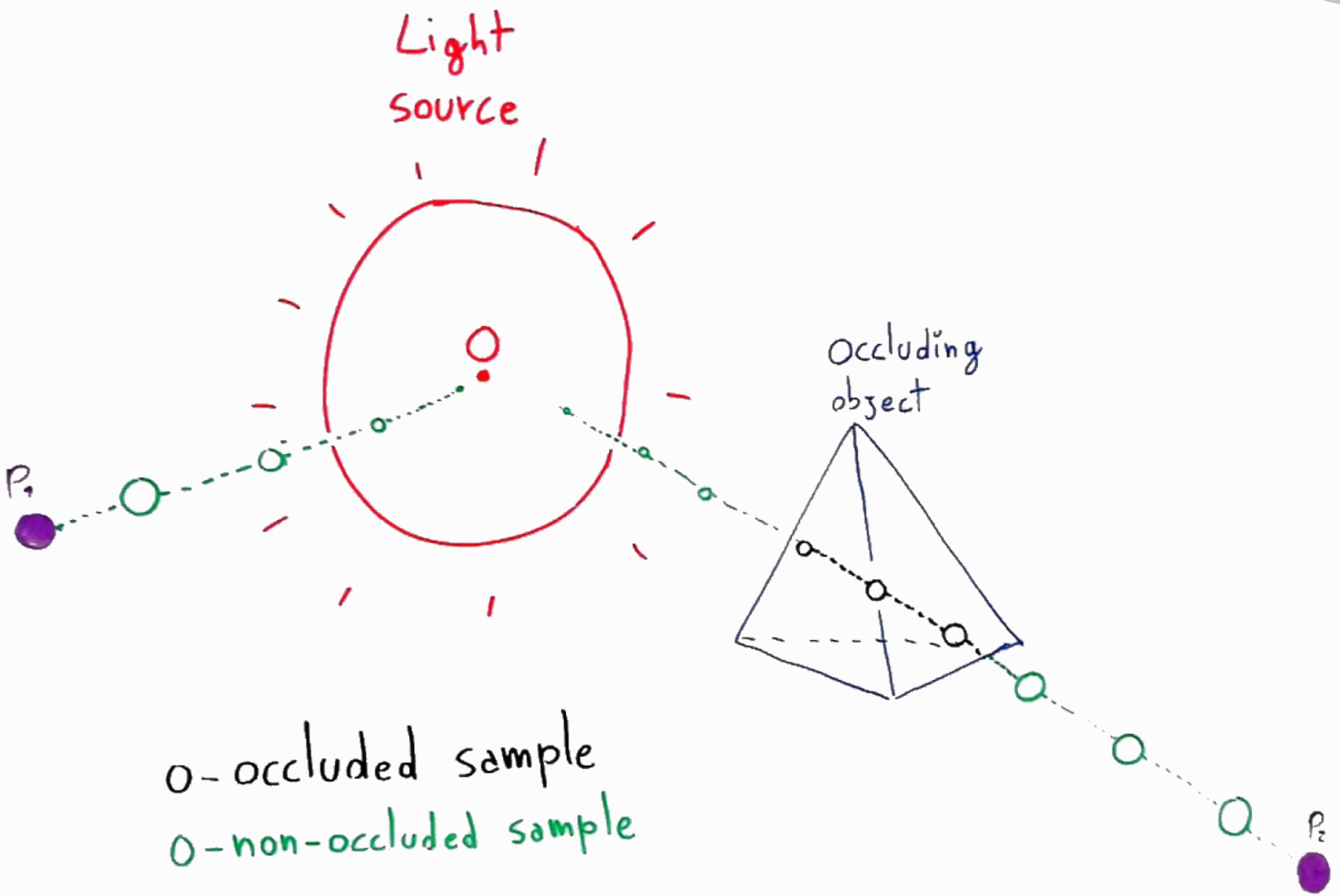
$$L(s,\theta,\phi) = exposure \cdot \sum_{i=0}^{n} \left(decay^{i} \cdot weight \cdot \frac{L(s_i,\theta_i)}{n} \right) \qquad (4)$$

where:

  - *exposure* controls the overall intensity of the postprocess
  - *weight* controls the intensity of each sample
  - $decay^{i}$ (with *decay* in range [0,1]) dissipates each sample's contributions as the ray progresses away from the object

- To compute the resulting illumination we sample the rendered scene (used as a texture) at regular intervals of length $density/n$

Explanatory Image



Implementation

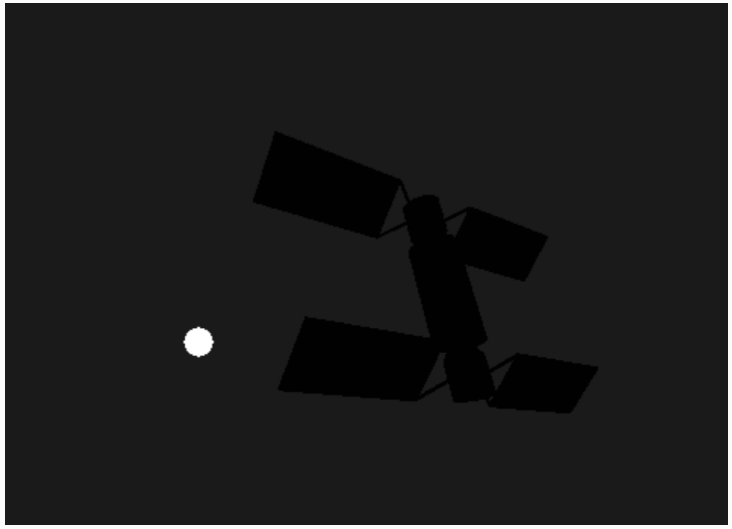
Technologies used

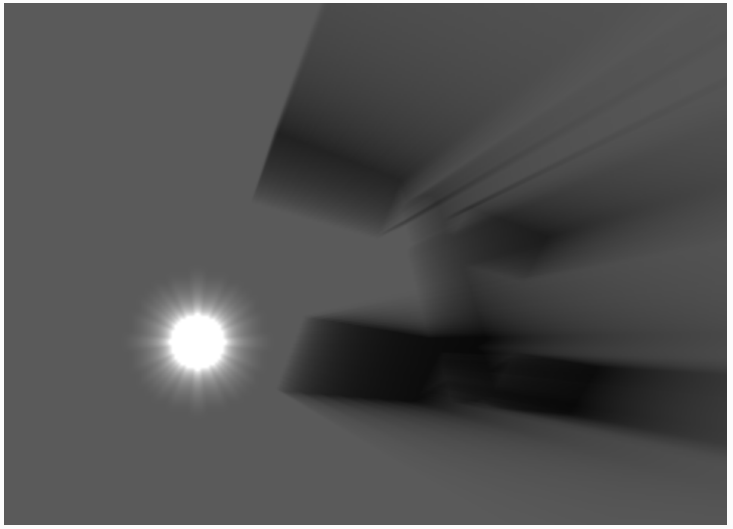
- `Typescript`, a typed superset of `Javascript`
- `Three.js`, a cross-browser `Javascript` library used to create and display 3D computer graphics based on `WebGL`
- `Webpack`, an open-source `Javascript` module bundler

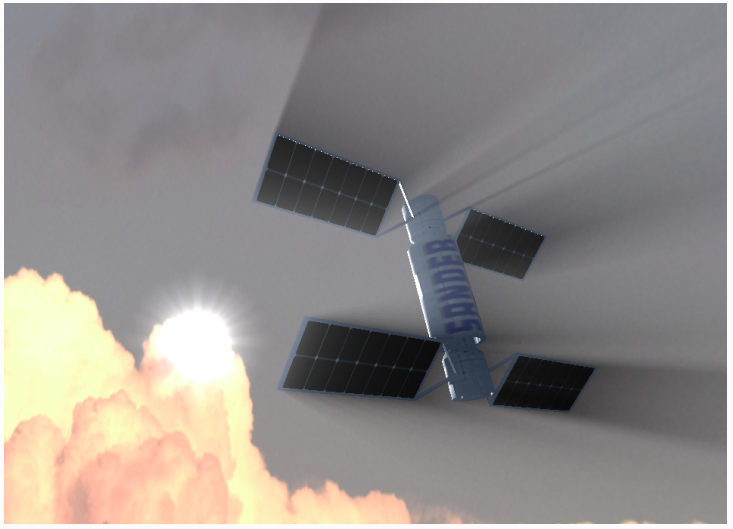
Effect Composers

- `EffectComposers` included in `three.js` provide a convenient API to chain postprocessing passes
- Abstraction layer over framebuffer objects
- Two `EffectComposers` used:
  - An *occlusion* `EffectComposer` that renders the light scattering effect
  - A *scene* `EffectComposer` that blends the scattering effect with the original rendered scene, and its result is rendered to screen
- Multiple `three.js` layers to manage object assignment to different effect composers

Operating Scheme







FLORENTINA STUDIORUM UNIVERSITAS

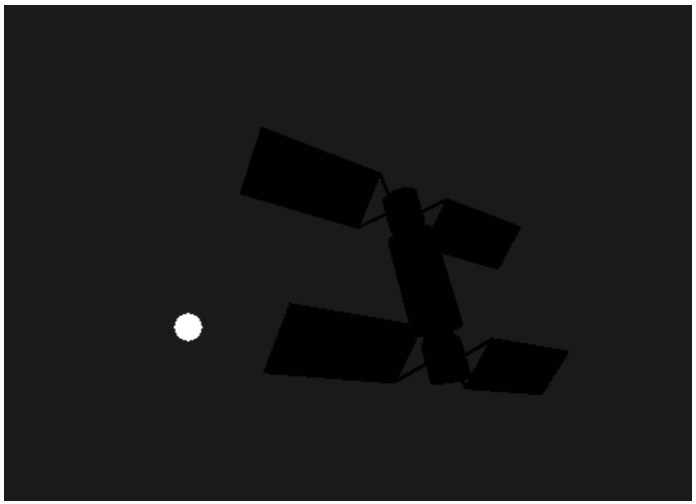
Occlusion Effect Composer

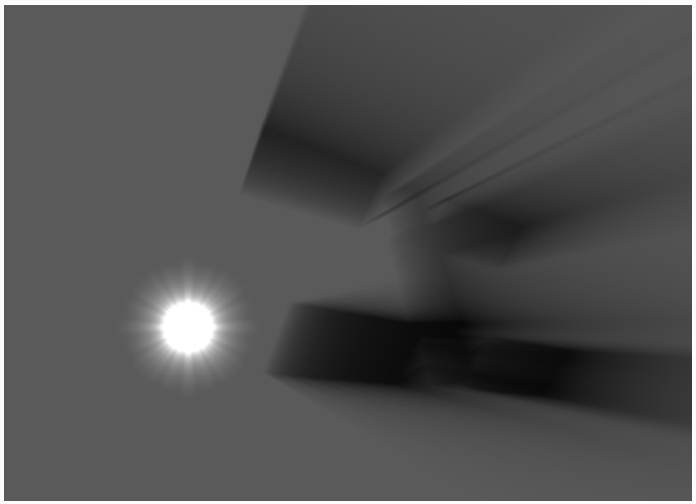

```
let occlusionRenderTarget = new WebGLRenderTarget(
    window.innerWidth * this.lightPassScale,
    window.innerHeight * this.lightPassScale);

let occlusionComposer = new EffectComposer(renderer, occlusionRenderTarget);
occlusionComposer.addPass(new RenderPass(this.scene, this.camera));

let scatteringPass = new ShaderPass(occlusionShader);
this.shaderUniforms = scatteringPass.uniforms;
occlusionComposer.addPass(scatteringPass);

let finalPass = new ShaderPass(CopyShader);
occlusionComposer.addPass(finalPass);
```





Occlusion Effect Composer

Volumetric scattering vertex shader

- The *scattering pass* is the focus of the postproccesing chain we implemented
- The vertex shader is a *pass through* shader that doesn't affect the scene geometry

```
1 out vec2 vUv;
2
3 void main(){
4     vUv = uv;
5     gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
6 }
```

- uv represents per-vertex texture coordinates of the previous pass

Occlusion Effect Composer

Volumetric scattering fragment shader


```glsl
uniform sampler2D tDiffuse;
uniform vec2 lightPosition;
uniform float decay;
uniform float exposure;
uniform int samples;
uniform float weight;
uniform float density;

in vec2 vUv;
out vec4 fragColor;

void main() {
    vec2 ray = vUv - lightPosition;
    vec2 delta = ray * (1. / float(samples)) * density;
    vec4 color = texture(tDiffuse, vUv);
    vec2 currentPos = vUv;
    float illuminationDecay = 1.;

    for (int i = 1; i < samples; ++i) {
        currentPos -= delta;
        vec4 currentColor = texture(tDiffuse, currentPos);
        illuminationDecay *= decay;
        currentColor *= illuminationDecay * weight;
        color += currentColor;
    }
    fragColor = color * exposure;
}
```

Occlusion Effect Composer

Volumetric scattering fragment shader


```
13  vec2 ray = vUv - lightPosition;
14  vec2 delta = ray * (1. / float(samples)) * density;
15  vec4 color = texture(tDiffuse, vUv);
16  vec2 currentPos = vUv;
17  float illuminationDecay = 1.;
```

- vUv parameter is the texture coordinate passed from the vertex shader
- ray represents the vector from the light position in the screen to the current pixel position
- currentPos will be updated in a for loop and represents the current sample position

Occlusion Effect Composer

Volumetric scattering fragment shader

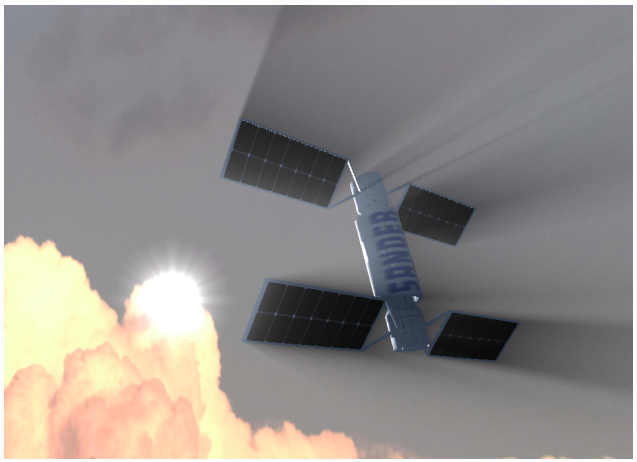

```
19  for (int i = 1; i < samples; ++i) {
20      currentPos -= delta;
21      vec4 currentColor = texture(tDiffuse, currentPos);
22      illuminationDecay *= decay;
23      currentColor *= illuminationDecay * weight;
24      color += currentColor;
25  }
26
27  fragColor = color * exposure;
```

- Loop implements the equation

$$L(s,\theta,\phi) = exposure \cdot \sum_{i=0}^{n} \left(decay^{i} \cdot weight \cdot \frac{L(s_i,\theta_i)}{n} \right)$$

Scene Effect Composer


```
1  let sceneComposer = new EffectComposer(renderer);
2  let renderPass = new RenderPass(this.scene, this.camera)
3  sceneComposer.addPass(renderPass);
4
5  let blendingPass = new ShaderPass(blendingShader);
6  blendingPass.uniforms.tOcclusion.value =
   ↪  occlusionRenderTarget.texture;
7
8  blendingPass.renderToScreen = true;
9  sceneComposer.addPass(blendingPass);
```



Scene Effect Composer

Blending Fragment Shader

- The `blendingPass` blends the basic scene with the `occlusionComposer` output scene
- The vertex shader is the same *pass through* shader of the occlusion composer

The fragment shader implementation is:

```
1  uniform sampler2D tDiffuse;
2  uniform sampler2D tOcclusion;
3
4  in vec2 vUv;
5  out vec4 fragColor;
6
7  void main() {
8      vec4 originalColor = texture(tDiffuse, vUv);
9      vec4 blendingColor = texture(tOcclusion, vUv);
10     fragColor = originalColor + blendingColor;
11 }
```

Rendering


```
1  public render() {
2      this.controls.update();
3      updateShaderLightPosition(this.lightSphere, this.camera,
       ↪  this.shaderUniforms)
4
5      this.camera.layers.set(OCCLUSION_LAYER);
6      renderer.setClearColor("#111111")
7      this.occlusionComposer.render();
8
9      this.camera.layers.set(DEFAULT_LAYER);
10     renderer.setClearColor("#000000");
11     this.sceneComposer.render();
12
13 }
```

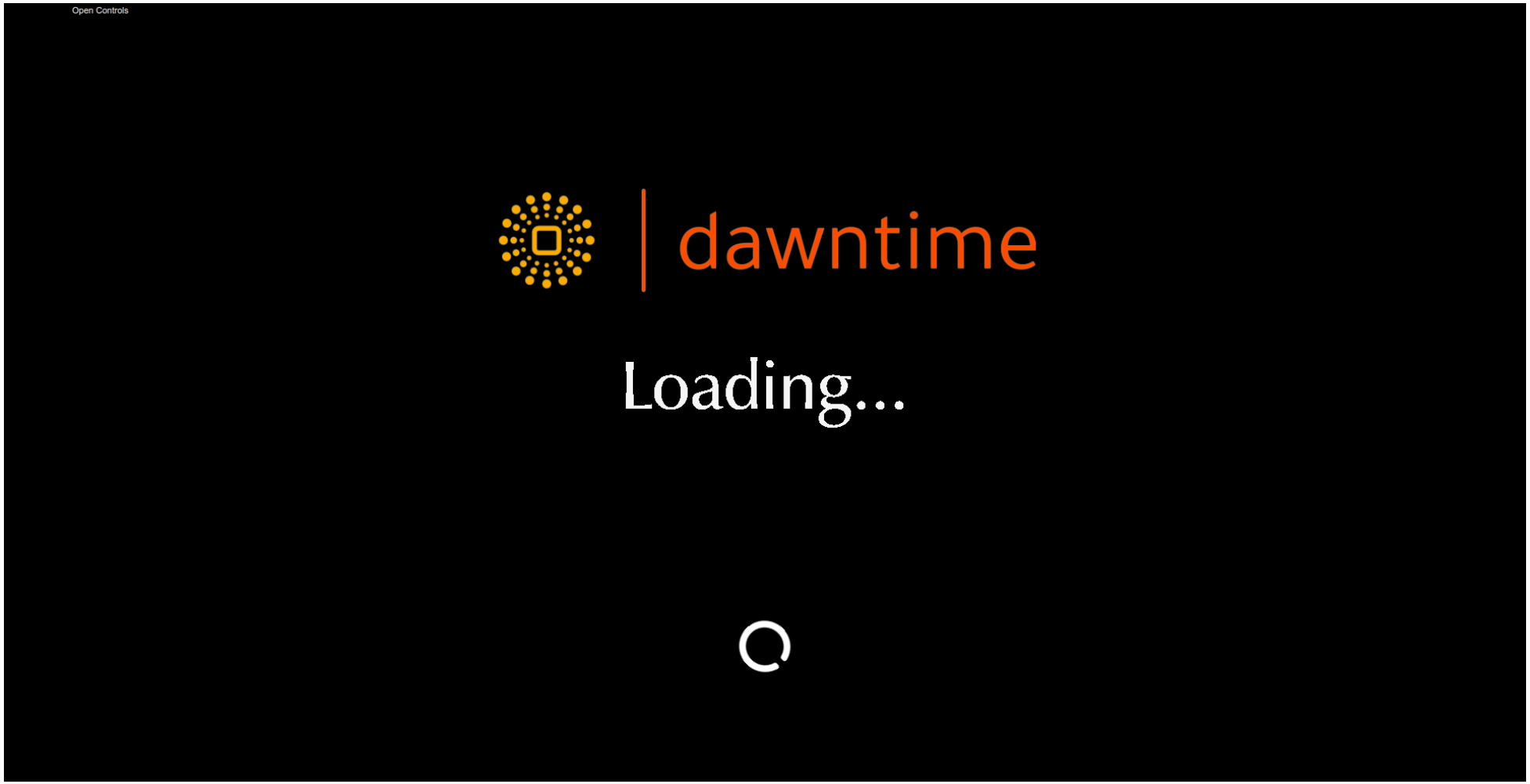
Light Position Updating

- We need to manually convert the uniform variable linked to the position of the light from normalized device coordinates to texture coordinates

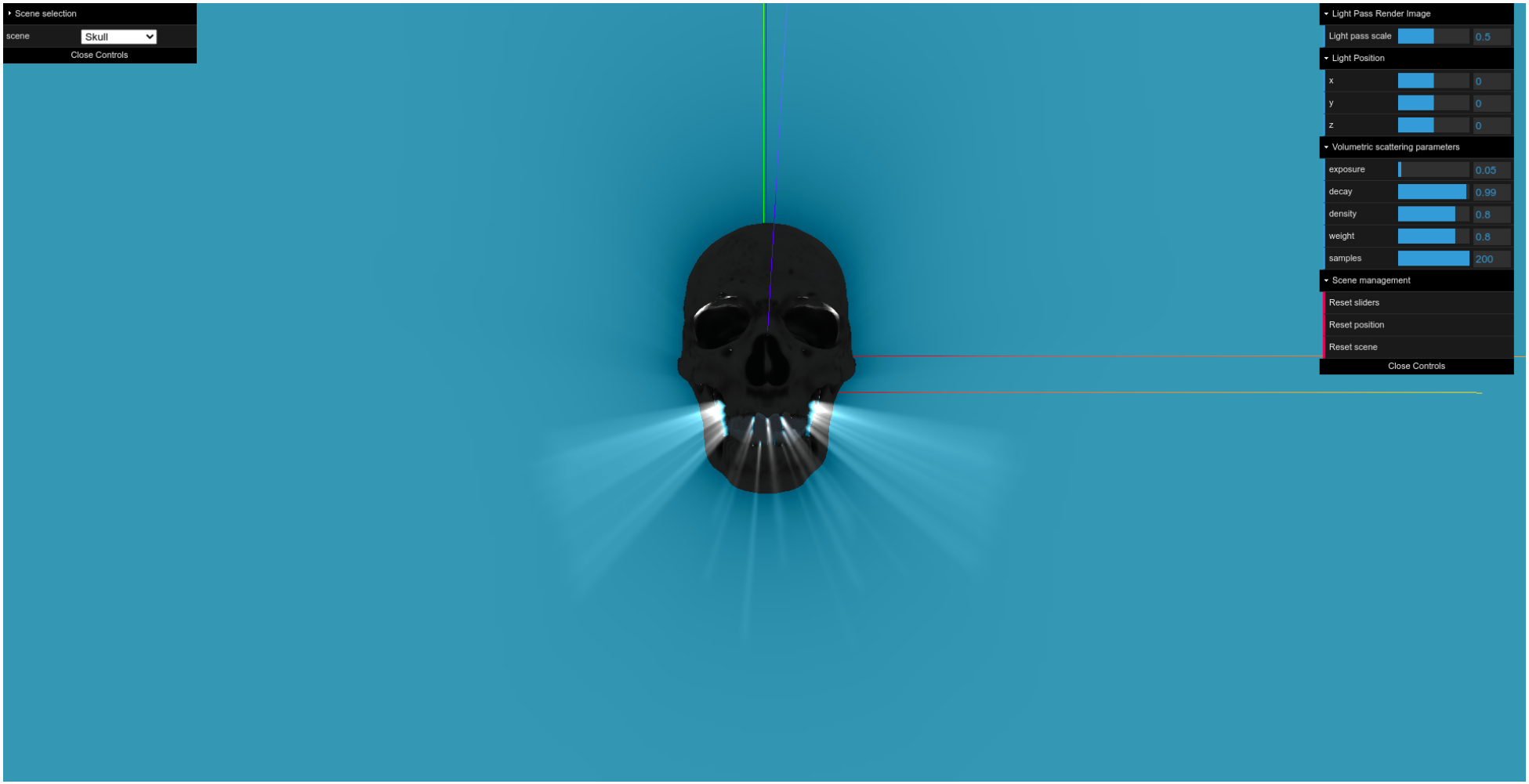
```
1  function updateShaderLightPosition(lightSphere: Mesh,
2      camera: Camera,
3      shaderUniforms: any) {
4
5      let screenPosition =
       ↪  lightSphere.position.clone().project(camera);
6
7      let newX = 0.5 * (screenPosition.x + 1);
8      let newY = 0.5 * (screenPosition.y + 1);
9
10     shaderUniforms.lightPosition.value.set(newX, newY);
11 }
```

Demo Images

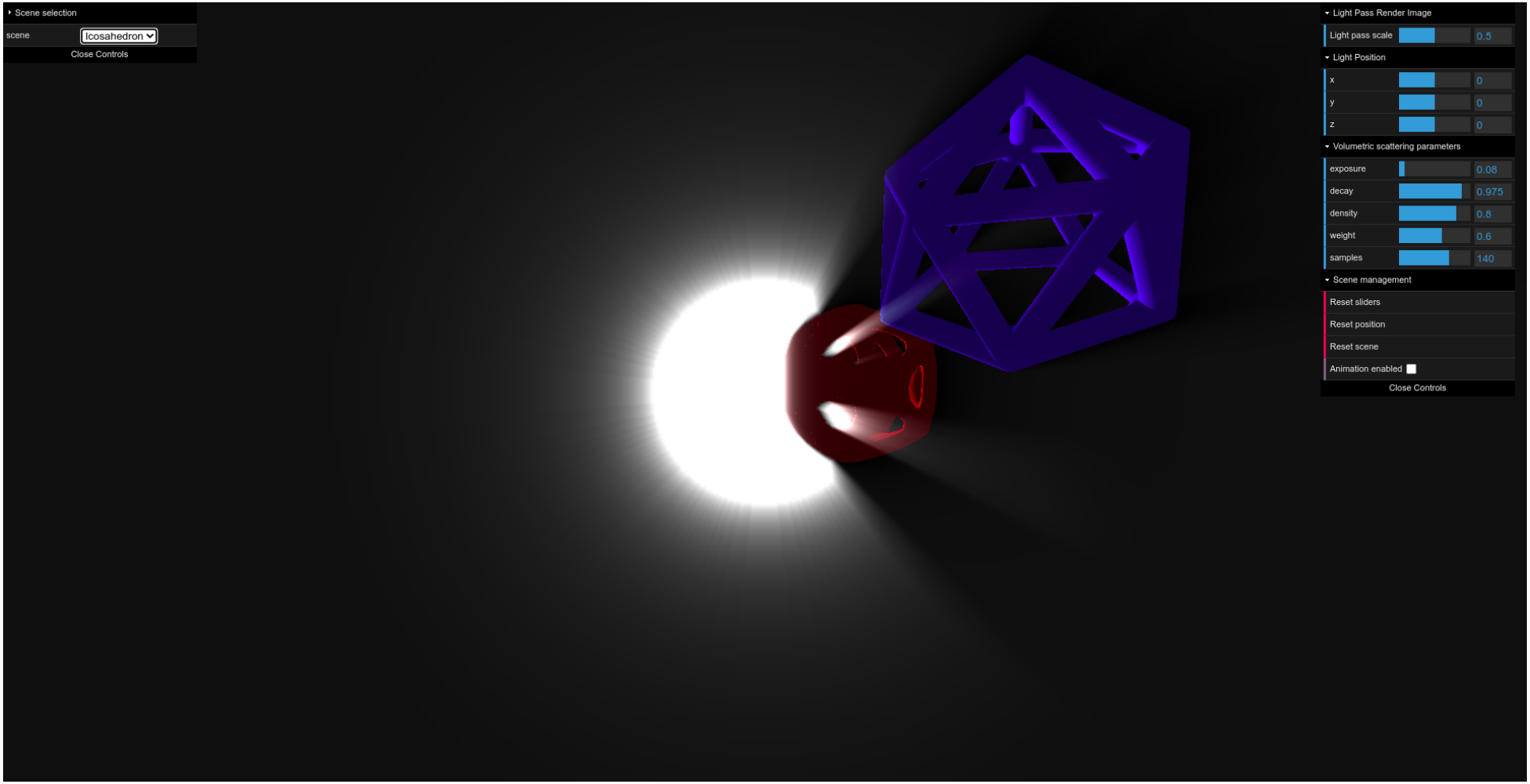
Loading Screen



Skull Scene


Scene selection
scene
Skull
Close Controls
Light Pass Render Image
Light pass scale
0.5
Light Position
x
0
y
0
z
0
Volumetric scattering parameters
exposure
0.05
decay
0.99
density
0.8
weight
0.8
samples
200
Scene management
Reset sliders
Reset position
Reset scene
Close Controls

Icosahedron Scene

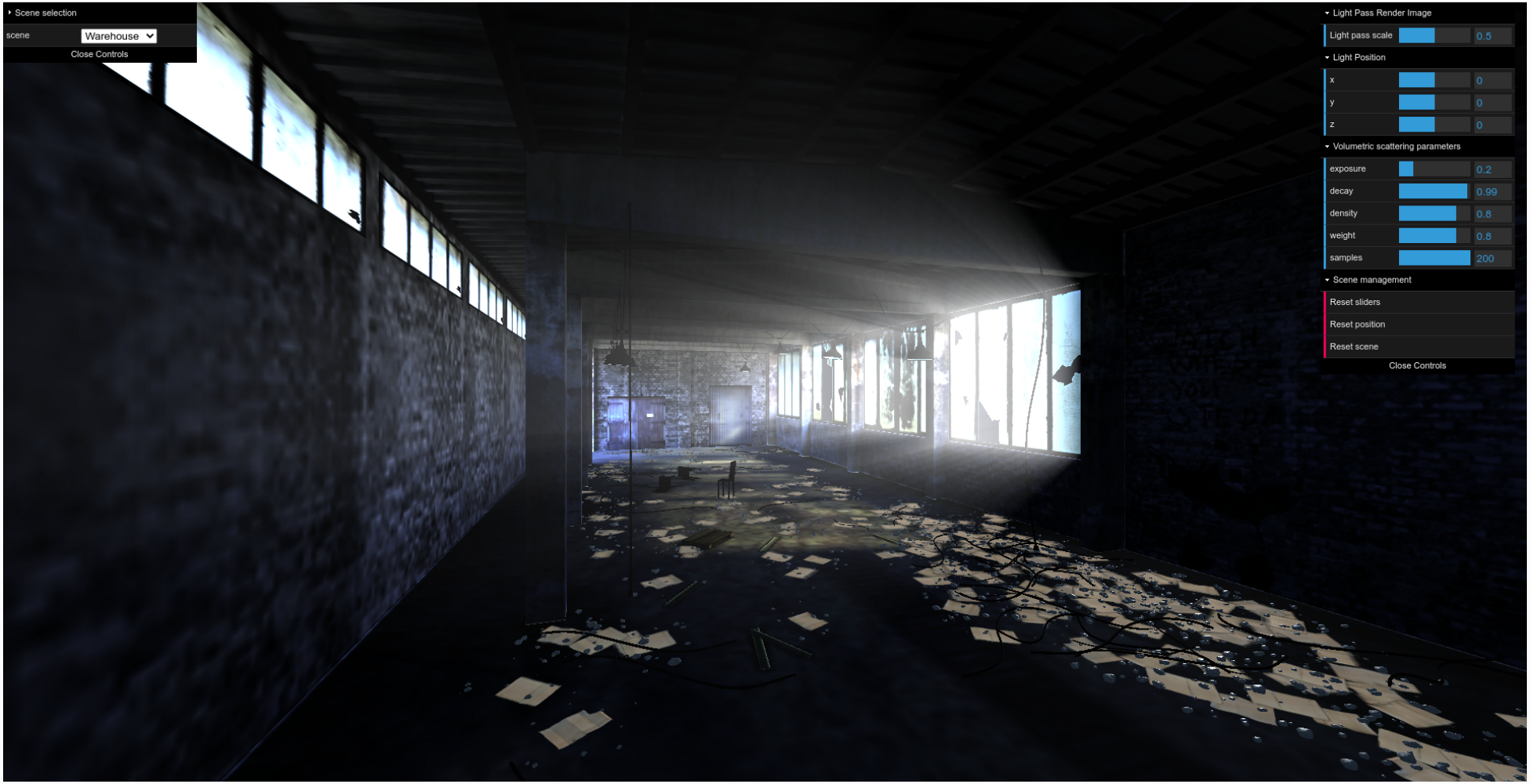


Ship Scene


Scene selection
scene
Ship
Close Controls
Light Pass Render Image
Light pass scale
0.5
Light Position
x
-1.8
y
0
z
-22
Volumetric scattering parameters
exposure
0.05
decay
0.99
density
0.8
weight
0.8
samples
200
Scene management
Reset sliders
Reset position
Reset scene
Animation enabled
Close Controls

Satellite Scene

Warehouse Scene




demo available here

FLORENTINA STUDIORUM