UNIVERSITÀ DEGLI STUDI FIRENZE

Implementation of Crepuscular Rays using Three.js

Lorenzo Gianassi, Francesco Gigli

Computer Graphics and 3D Course Project held by Prof. Stefano Berretti

Introduction

Theory Introduction

- **Crepuscular rays** or ”God rays” are sunbeams that originate when the sun is below the horizon, during twilight hours

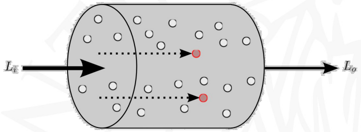
- Crepuscular rays are noticeable when the contrast between light and dark is most obvious

UNIVERSITÀ DEGLI STUDI FIRENZE

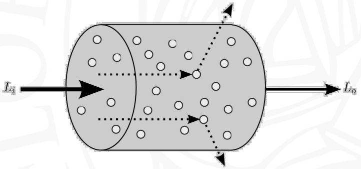
Light Scattering

Processes that affect the distribution of radiance in an environment with participating media:

- **Absorption:** the reduction in radiance due to the conversion of light to another form of energy

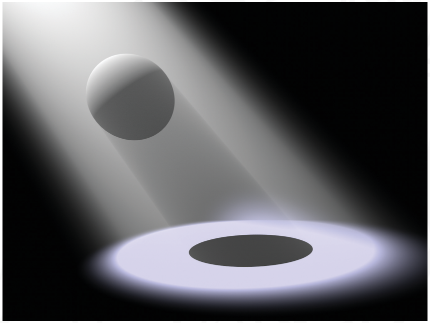


- **Scattering:** radiance heading in one direction that is scattered to other directions due to collisions with particles



Volumetric Light Scattering

- **Volumetric Light Scattering** is an effect where a spotlight shining through a participating medium illuminates particles in the medium and casts a volumetric shadow



- This project implements this effect based on the postprocessing technique described by Kenny Mitchell

Analytic Model

To calculate the illumination at each pixel, we must account for scattering from the light source to that pixel and whether or not the scattering media is occluded.

$$L(s,\theta) = L_0 e^{-\beta_\alpha S} + \frac{1}{\beta_{\mathrm{ex}}} E_{\mathrm{sun}} \beta_{\mathrm{sc}}(\theta) \left(1 - e^{-\beta_\alpha S}\right) \qquad (1)$$

- The first term calculates the amount of light absorbed from the point of emission to the viewpoint
- The second term calculates the additive amount due to light scattering into the path of the view ray

$$L(s,\theta) = L_0 e^{-\beta_\alpha S} + \frac{1}{\beta_\alpha} E_{\text{sun}}\, \beta_{\text{sc}}(\theta) \left(1 - e^{-\beta_\alpha S}\right)$$

Where:

- L_0 is the illumination from the Light source
- s is the distance traveled through the media is the angle between the ray and the sun
- θ is the angle between the ray and the sun
- β_{ex} is the extinction constant composed of light absorption and out-scattering properties
- E_{sun} is the source illumination from the sun
- β_{sc} is the angular scattering term composed of Rayleigh and Mie scattering properties.

Occlusion probability

The effect due to occluding matter is modeled here simply as an attenuation of the source illumination.
Recall the following:

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

- D() is the combined attenuated sun-occluding objects' opacity for the view location.

- We can estimate the probability of occlusion at each pixel by summing samples along a ray to the light source in image space.
- The proportion of samples that hit the emissive region versus those that strike occluders gives us the desired percentage of occlusion, D().

If we divide by the number of samples, n, the post-process simply resolves to an additive sampling of the image:

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

UNIVERSITÀ DEGLI STUDI FIRENZE

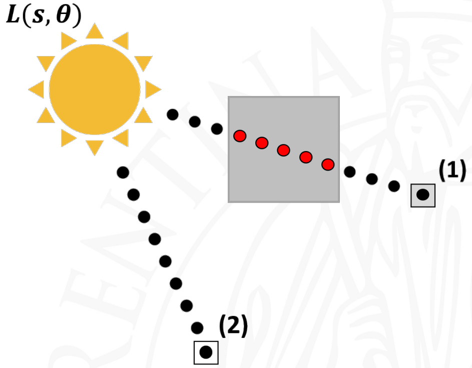
Attenuation Coefficients

In addition, we introduce attenuation coefficients to parameterize control of the summation:

$$L(s,\theta,\phi) = \text{ exposure } \times \sum_{i=0}^{n} \text{ decay }^{i} \times \text{ weight } \times \frac{L(s_i,\theta_i)}{n} \quad (4)$$

- *Exposure* controls the overall intensity of the post-process
- *Weight* controls the intensity of each sample
- *Decay*^{i} (for the range [0, 1]) dissipates each sample's contribution as the ray progresses away from the light source

Ray Casting



Each dot represents a sample:

- (1) have an occluder, darker
- (2) doesn’t have an occluder, brighter

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Implementation

Implementation

The implementation of the project was made by:

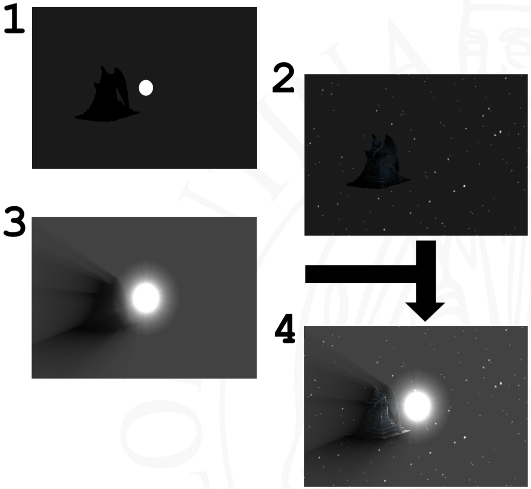
- **Javascript**
- **Three.js**: Javascript library used to create and display 3D computer graphics based on WebGL
- **Webpack**: used as *Module Bundler*

Effect Composers

- We used the **Effect Composers** to implement post-processing effects in Three.js.
- The class manages a chain of post-processing passes to produce the final visual result.
- Two Effect Composers:
    - **Occlusion Composer**: renders the *Light Scattering* Effect
    - **Scene Composer**: blends the original rendered scene with the *Scattering Effect*, to afterwards render the result
- Two layers (provided by Three.js) we're used to menage the the assignment of the objects to the Effect Composeres

UNIVERSITÀ DEGLI STUDI FIRENZE

Process Pipeline



Occlusion Composers

```
// A preconfigured render target internally used by EffectComposer.
let target = new THREE.WebGLRenderTarget(window.innerWidth / 2, window.innerHeight / 2,
        renderTargetParameters)

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

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

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

Pass Through Vertex Shader

- The Scattering Pass is the focus of our postprocessing chain

```
varying vec2 vUv;

void main(){
    vUv = uv;
    gl_Position = projectionMatrix * modelViewMatrix *
        vec4(position, 1.0);
}
```

- uv is the per-vertex texture coordinates coming from the previous pass

Scattering Fragment Shader

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

varying vec2 vUv;

void main() {
    vec2 delta = vUv − lightPosition ;
    delta *= (1.0 / float (samples)) * density;
    vec4 color = texture( tDiffuse , vUv);
    vec2 currentPos = vUv;
    float illuminationDecay = 1.0;

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

    gl_FragColor = color * exposure;
}
```

Scattering Fragment Shader

```
1   void main() {
2       vec2 delta = vUv - lightPosition;
3       delta *= (1.0 / float(samples)) * density;
4       vec4 color = texture(tDiffuse, vUv);
5       vec2 currentPos = vUv;
6       float illuminationDecay = 1.0;
7       ...
```

- vUv represents the texture coordinate coming from the vertex shader
- delta represents the vector from the light position in the screen to the current pixel position
- delta is then updated as seen in line 3 the currentPos represents the current sample position that is updated inside the for loop

Scattering Fragment Shader

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

    gl_FragColor = color * exposure;
```

The *for loop* implements the equation:

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

Scene Composer

```
// Scene Composer
let sceneComposer = new EffectComposer(renderer);
sceneComposer.addPass(new RenderPass(this.scene,
    this.camera));

//Blending Pass
let blendingPass = new ShaderPass(blendingShader);
blendingPass.uniforms.tOcclusion.value =
    target.texture;

sceneComposer.addPass(blendingPass);
```

Blending Fragment Shader

```
uniform sampler2D tDiffuse;
uniform sampler2D tOcclusion;

varying vec2 vUv;

void main() {
    vec4 originalColor = texture(tDiffuse, vUv);
    vec4 blendingColor = texture(tOcclusion, vUv);
    gl_FragColor = originalColor + blendingColor;
}
```

- The Vertex Shaders used is the same used by the Occlusion Composer (Pass through Vertex Shader)
- The **Blending Pass** blends the output scene of the *Occlusion Composer* and the *Basic Scene*

Render

```
    render() {
    this.controls.update();

    this.camera.layers.set(OCCLUSION_LAYER);
    renderer.setClearColor("#1a1a1a")

    this.occlusionComposer.render();
    this.camera.layers.set(DEFAULT_LAYER);
    renderer.setClearColor("#000000");

    this.sceneComposer.render();
}
```

Update Shader Light Position

```
function updateShaderLightPosition(lightSphere,
    camera, shaderUniforms){
let screenPosition =
    lightSphere.position.clone().project(camera);
let newX = 0.5 * (screenPosition.x + 1);
let newY = 0.5 * (screenPosition.y + 1);
shaderUniforms.lightPosition.value.set(newX, newY)
}
```

Build Light

```
// Light management
buildLight(radius, width, height, x, y, z, sunColor) {
// AmbientLight
this.ambientLight = new THREE.AmbientLight("#2c3e50");
this.scene.add(this.ambientLight);


// PointLight
this.pointLight = new THREE.PointLight("#ffffff");
this.scene.add(this.pointLight);

// Geometry and Material
let geometry = new THREE.SphereBufferGeometry(radius, width, height);
let material = new THREE.MeshBasicMaterial({ color: sunColor });
this.lightSphere = new THREE.Mesh(geometry, material);
this.lightSphere.layers.set(OCCLUSION_LAYER);
this.lightSphere.position.set(x,y,z);


this.scene.add(this.lightSphere);
}
```

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Build Light

```
buildScene() {
loader.load(statueFile , gltf => {
      gltf .scene.traverse(function (obj) {
           if (obj.isMesh) {
                let material = new THREE.MeshBasicMaterial({ color: "#000000" });
                let occlusionObject = new THREE.Mesh(obj.geometry, material)
                occlusionObject.layers.set(OCCLUSION_LAYER)
                if (obj.parent != null ) {
                     obj.parent.add(occlusionObject)
                }

           }
      })

      this .scene.add(gltf.scene);
      gltf .scene.position.copy(this.groupBasePosition);

}, function (error) {
      console.error(error);
});

this .camera.position.copy(this.baseCameraPosition);
this .controls.update();
this .buildBackGround(galaxy, 80, 64, 64)
}
```

UNIVERSITÀ DEGLI STUDI FIRENZE

Live-Demo is available here