A Computational Study on Mie Scattering in Nanoparticles

Hennie Kotze

June 10, 2024

Abstract

This paper discusses a computational study on the rendering of Mie scattering in nanoparticles (sizes comparable with the wavelengths of visible light) using a custom-built Java ray tracing engine.

The engine applies the principles of electromagnetic wave scattering and geometric optics to simulate and visualize the results of interactions of light with nanoparticles.

Contents

1 Introduction

The refraction of visible light through everyday transparent objects with sufficiently high refractive indices – such as lenses, glass marbles or water-filled containers – is a relatively common sight, and the manner in which the light is refracted through these transparent object is familiar.

Nanoparticles, on the other hand, exhibit unique optical properties when interacting with light, which is fundamentally governed by the principles of Mie scattering. This study aims to accurately simulate these interactions using a custom-built ray tracing engine written in Java. The engine models the behaviour of light as it encounters nanoparticles, focusing on rendering visualizations of the effects of scattering, providing a better understanding of the underlying physics. This paper presents the mathematical formulations and computational techniques used in the simulation, as well as the resulting visualizations.

2 Mathematical and Physical Background

2.1 Mie Scattering

Mie scattering is a form of light scattering that describes the interaction of electromagnetic waves with spherical particles. It is governed by Maxwell's equations and is applicable when the particle size is comparable to the wavelength of the incident light.

The key parameter in Mie scattering is the size parameter x, which is defined as:

$$x = \frac{2\pi r}{\lambda} \tag{1}$$

where r is the radius of the particle and λ is the wavelength of the incident light.

2.1.1 Bessel Functions

The spherical Bessel functions of the first kind, $j_n(x)$, and the spherical Hankel functions of the first kind, $h_n^{(1)}(x)$, play crucial roles in the formulation of Mie scattering. These functions are defined as:

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+\frac{1}{2}}(x) \tag{2}$$

$$h_n^{(1)}(x) = j_n(x) + iy_n(x) \tag{3}$$

where $J_{n+\frac{1}{2}}(x)$ is the ordinary Bessel function of the first kind, and $y_n(x)$ is the spherical Neumann function.

The derivatives of the spherical Bessel functions are computed numerically using the central difference method:

$$j_n'(x) \approx \frac{j_n(x+H) - j_n(x-H)}{2H} \tag{4}$$

where H is a small perturbation.

2.1.2 Mie Coefficients

The Mie coefficients a_n and b_n are calculated as follows:

$$a_n = \frac{m\psi_n(mx)\psi_n'(x) - \psi_n(x)\psi_n'(mx)}{m\psi_n(mx)\xi_n'(x) - \xi_n(x)\psi_n'(mx)} \tag{5}$$

$$b_n = \frac{\psi_n(mx)\psi_n'(x) - m\psi_n(x)\psi_n'(mx)}{\psi_n(mx)\xi_n'(x) - m\xi_n(x)\psi_n'(mx)} \tag{6}$$

where $\psi_n(x) = xj_n(x)$ and $\xi_n(x) = xh_n^{(1)}(x)$, and m is the complex refractive index of the particle:

$$m = m_r + im_i \tag{7}$$

2.1.3 Scattering and Absorption Efficiencies

The scattering efficiency Q_{sca} and the absorption efficiency Q_{abs} are given by:

$$Q_{sca} = \frac{2}{x^2} \sum_{n=1}^{\infty} (2n+1) \left(|a_n|^2 + |b_n|^2 \right) \tag{8}$$

$$Q_{abs} = \frac{2}{x^2} \sum_{n=1}^{\infty} (2n+1) \left(\mathrm{Re}(a_n + b_n) - |a_n|^2 - |b_n|^2 \right) \tag{9}$$

The total extinction efficiency Q_{ext} is the sum of the scattering and absorption efficiencies:

$$Q_{ext} = Q_{sca} + Q_{abs} \tag{10}$$

2.1.4 Phase Function

The phase function $P(\cos\theta)$ describes the angular distribution of scattered light. For Mie scattering, the Henyey-Greenstein phase function is often used:

$$P(\cos\theta) = \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}} \tag{11}$$

where g is the asymmetry parameter, typically taken to be around 0.9 for spherical particles.

2.2 Implementation in Code

The Mie scattering calculations are implemented in the `MieScattering` class. This class handles the computation of the size parameter, Bessel functions, Mie coefficients, and efficiencies. The implementation relies on the Apache Commons Math library[1], which provides robust numerical methods and special functions. `MieScattering` uses the `BesselJ` class from the library which also includes methods for computing the spherical Hankel functions and their derivatives.

2.2.1 Class Constructor

The constructor initializes the particle radius, wavelength, and complex refractive index:

```
public MieScattering(double radius, double wavelength, double refractiveIndexReal, double refractiveIndexImag) {
    this.radius = Math.max(1e-9, radius); // Avoid too small radius
    this.wavelength = Math.max(1e-9, wavelength); // Avoid too small wavelength
    this.refractiveIndex = new Complex(refractiveIndexReal, refractiveIndexImag);
    this.sizeParameter = 2 * Math.PI * this.radius / this.wavelength;
}
```

2.2.2 Size Parameter Calculation

The size parameter x is computed as:

$$x = 2\pi \frac{r}{\lambda} \tag{12}$$

2.2.3 Spherical Bessel Functions

The spherical Bessel functions and their derivatives are computed using:

```
private double sphericalBesselJ(int n, double x) {
    if (x == 0) return 0;
    return Math.sqrt(Math.PI / (2 * x)) * new BesselJ(n + 0.5).value(x);
}

private double sphericalBesselJPrime(int n, double x) {
    return (sphericalBesselJ(n, x + H) - sphericalBesselJ(n, x - H)) / (2 * H);
}
```

[1] Version 3.6.1 of the Apache Commons Math3 library is used in this software.

2.2.4 Mie Coefficients Calculation

The Mie coefficients a_n and b_n are calculated as:

```
private Complex[] calculateMieCoefficients(int n) {
    double x = getSizeParameter();
    if (x == 0) return new Complex[]{Complex.ZERO, Complex.ZERO};
    Complex m = refractiveIndex;

    double psi_n_x = sphericalBesselJ(n, x);
    double psi_n_x_prime = sphericalBesselJPrime(n, x);
    Complex psi_n_mx = new Complex(sphericalBesselJ(n, m.getReal() * x), 0);
    Complex psi_n_mx_prime = new Complex(sphericalBesselJPrime(n, m.getReal() * x), 0);
    Complex xi_n_x = sphericalHankel(n, x);
    Complex xi_n_x_prime = sphericalHankelPrime(n, x);

    if (xi_n_x_prime.equals(Complex.ZERO)) return new Complex[]{Complex.ZERO, Complex.ZERO};

    Complex numeratorA = m.multiply(psi_n_mx).multiply(psi_n_x_prime).subtract(new Complex(psi_n_x, 0).multiply(psi_n_mx_prime));
    Complex denominatorA = m.multiply(psi_n_mx).multiply(xi_n_x_prime).subtract(xi_n_x.multiply(psi_n_mx_prime));
    Complex a_n = numeratorA.divide(denominatorA);

    Complex numeratorB = psi_n_mx.multiply(psi_n_x_prime).subtract(m.multiply(new Complex(psi_n_x, 0)).multiply(psi_n_mx_prime));
    Complex denominatorB = psi_n_mx.multiply(xi_n_x_prime).subtract(m.multiply(xi_n_x).multiply(psi_n_mx_prime));
    Complex b_n = numeratorB.divide(denominatorB);

    return new Complex[]{a_n, b_n};
}
```

2.2.5 Efficiencies Calculation

The scattering and absorption efficiencies are computed as:

```
public double calculateScatteringEfficiency() {
    double x = getSizeParameter();
    if (x > 1e3) {
        System.err.println("Warning: Size parameter x is very large, results may not be accurate.");
        return 0; // Return 0 or some default value to avoid computation
    }
    double Q_sca = 0;
    for (int n = 1; n <= 100; n++) {
        Complex[] coefficients = calculateMieCoefficients(n);
        double anAbs = coefficients[0].abs();
        double bnAbs = coefficients[1].abs();
        Q_sca += (2 * n + 1) * (anAbs * anAbs + bnAbs * bnAbs);
    }
    double scatteringEfficiency = (2 / (x * x)) * Q_sca;
    return scatteringEfficiency;
}

public double calculateAbsorptionEfficiency() {
    double x = getSizeParameter();
    if (x > 1e3) {
        System.err.println("Warning: Size parameter x is very large, results may not be accurate.");
        return 0; // Return 0 or some default value to avoid computation
    }
    double Q_abs = 0;
    for (int n = 1; n <= 100; n++) {
        Complex[] coefficients = calculateMieCoefficients(n);
        double anReal = coefficients[0].getReal();
        double bnReal = coefficients[1].getReal();
        double anAbs = coefficients[0].abs();
        double bnAbs = coefficients[1].abs();
        Q_abs += (2 * n + 1) * (anReal + bnReal - anAbs * anAbs - bnAbs * bnAbs);
    }
    double absorptionEfficiency = (2 / (x * x)) * Q_abs;
    return absorptionEfficiency;
}
```

2.2.6 Phase Function Calculation

The phase function is computed as:

```
public double calculatePhaseFunction(double cosTheta) {
    double g = 0.9;
    double phaseFunction = (1 - g * g) / Math.pow(1 + g * g - 2 * g * cosTheta, 1.5);
    return phaseFunction;
}
```

3 Application of Mie Scattering in Rendering Techniques

3.1 Overview of Rendering Approach

The rendering process involves the following key steps:

1. **Ray Tracing**: Simulating the path of light rays as they interact with the particles.
2. **Mie Scattering Calculation**: Computing the scattering and absorption efficiencies for different wavelengths of light.
3. **Color Composition**: Combining the scattered light intensities to form the final pixel color.

3.2 Ray Tracing

The ray tracing technique is employed to simulate the interaction of light rays with spherical particles. Each ray is traced from the camera through the scene, and its interactions with the particles are calculated. The ray tracing algorithm determines whether a ray hits a particle, and if so, computes the subsequent scattering and refraction of the ray.

The core of this process is encapsulated in the `MieTransparentSphere` class, which extends the `Sphere` class. When a ray intersects a particle, the class calculates the scattering and refraction effects based on Mie scattering principles.

3.2.1 World Setup

The simulation environment, or "world," consists of several components: the camera, the spherical particle, the directional light, and the atmosphere. Each of these elements is essential for accurately simulating the scattering and refraction of light.

Camera The camera is responsible for capturing the scene from a specific viewpoint. It is defined by its position $\mathbf{C}$, its target point $\mathbf{T}$, and an up vector $\mathbf{U}$ which helps to define the camera's orientation. The field of view (FOV) and the image dimensions (width and height) also characterize the camera setup.

$$\mathbf{C} = \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}, \quad \mathbf{U} = \begin{pmatrix} U_x \\ U_y \\ U_z \end{pmatrix} \tag{13}$$

The camera generates rays that pass through each pixel of the image plane. The direction of each ray $\mathbf{D}$ is computed based on the camera's orientation and the position of the pixel on the image plane.

Spherical (Nano) Particle The spherical particle, defined by its center $\mathbf{P}$ and radius r, is the primary object of interest in the scene. The particle's properties include its size, refractive index, and absorption coefficient, all of which influence the scattering and refraction of light.

$$\mathbf{P} = \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix}, \quad r \tag{14}$$

Directional Light The directional light is characterized by its direction vector $\mathbf{L}$ and its color $\mathbf{C}_{light}$, which includes the intensities of the red, green, and blue components. The light direction is normalized to ensure consistency in calculations.

$$\mathbf{L} = \begin{pmatrix} L_x \\ L_y \\ L_z \end{pmatrix}, \quad \mathbf{C}_{light} = \begin{pmatrix} C_{light,r} \\ C_{light,g} \\ C_{light,b} \end{pmatrix} \tag{15}$$

Atmosphere The atmosphere is included in the simulation to visualize the scattering effects more effectively. It is defined by its scattering coefficient β_s, absorption coefficient β_a, and the sampling interval Δs. The atmosphere affects the light as it travels through it, contributing to the overall color seen by the camera.

$$\beta_s, \quad \beta_a, \quad \Delta s \tag{16}$$

3.2.2 Ray-Sphere Intersection

The intersection of a ray with a sphere is a fundamental calculation in ray tracing. The equation of a sphere with center $\mathbf{C}$ and radius r is given by:

$$(\mathbf{P} - \mathbf{C}) \cdot (\mathbf{P} - \mathbf{C}) = r^2$$

A ray is defined by its origin $\mathbf{O}$ and direction $\mathbf{D}$:

$$\mathbf{P}(t) = \mathbf{O} + t\mathbf{D}$$

Substituting the ray equation into the sphere equation gives:

$$(\mathbf{O} + t\mathbf{D} - \mathbf{C}) \cdot (\mathbf{O} + t\mathbf{D} - \mathbf{C}) = r^2$$

This expands to a quadratic equation in t:

$$(\mathbf{D} \cdot \mathbf{D})t^2 + 2(\mathbf{O} - \mathbf{C}) \cdot \mathbf{D}t + (\mathbf{O} - \mathbf{C}) \cdot (\mathbf{O} - \mathbf{C}) - r^2 = 0$$

Solving this quadratic equation for t determines if and where the ray intersects the sphere. The discriminant Δ of the quadratic equation must be non-negative for a real intersection:

$$\Delta = (2(\mathbf{O} - \mathbf{C}) \cdot \mathbf{D})^2 - 4(\mathbf{D} \cdot \mathbf{D})((\mathbf{O} - \mathbf{C}) \cdot (\mathbf{O} - \mathbf{C}) - r^2)$$

If $\Delta \geq 0$, the ray intersects the sphere at:

$$t = \frac{-(\mathbf{O} - \mathbf{C}) \cdot \mathbf{D} \pm \sqrt{\Delta}}{\mathbf{D} \cdot \mathbf{D}}$$

3.2.3 Scattering and Refraction

Once an intersection is detected, the scattering and refraction of the ray are computed. Mie scattering principles are used to determine the scattering coefficients and phase functions for the RGB components of light.

The refractive index of the particle, m, affects the refraction of the incoming ray. Snell's law describes the relationship between the angles of incidence θ_i and refraction θ_t:

$$n_1 \sin\theta_i = n_2 \sin\theta_t$$

where n_1 and n_2 are the refractive indices of the media. The direction of the refracted ray is calculated using:

$$\mathbf{T} = \eta\mathbf{I} + (\eta \cos\theta_i - \sqrt{1 - \eta^2(1 - \cos^2\theta_i)})\mathbf{N}$$

where $\eta = \frac{n_1}{n_2}$, $\mathbf{I}$ is the incident ray direction, and $\mathbf{N}$ is the normal at the point of intersection.

The phase functions for the RGB components are given by:

$$P(\cos\theta) = \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{1.5}}$$

where g is the asymmetry parameter. The scattering intensities are calculated as:

$$I_{\text{scattered}} = P(\cos\theta) \times Q_{\text{scattering}} \times (1 - Q_{\text{absorption}})$$

These calculations are performed for each color component (red, green, blue) to determine the final color of the scattered light.

3.2.4 Ray Intersection Status

The ray intersection status is determined using the following logic:
- If the ray hits the sphere, the scattering and refraction calculations are performed.
- If the ray undergoes total internal reflection, it is marked as `INTERNAL_REFLECTION`.
- If the ray exits the sphere, it is marked as `EXITING`.

The final color of the pixel is determined by combining the scattered and refracted light intensities:

$$\text{Color}_{\text{final}} = \text{Color}_{\text{scattered}} + \text{Color}_{\text{refracted}}$$

This approach ensures that the Mie scattering principles are correctly applied in the rendering process, producing accurate and realistic visualizations of light interactions with spherical particles.

3.3 Mie Scattering Calculation

For each ray-particle intersection, the scattering and absorption efficiencies are computed using Mie theory. The scattering efficiency Q_{sca} and absorption efficiency Q_{abs} are calculated for red, green, and blue wavelengths. These efficiencies determine the amount of light scattered and absorbed by the particle.

The refractive index $m = n + ik$ of the particle is used to compute the Mie coefficients a_n and b_n for each wavelength. These coefficients are then used to calculate the scattering and absorption efficiencies.

$$Q_{\text{sca}} = \frac{2}{x^2} \sum_{n=1}^{\infty} (2n+1) \left(|a_n|^2 + |b_n|^2 \right) \tag{17}$$

$$Q_{\text{abs}} = \frac{2}{x^2} \sum_{n=1}^{\infty} (2n+1) \left(\text{Re}(a_n + b_n) - |a_n|^2 - |b_n|^2 \right) \tag{18}$$

These efficiencies are precomputed for the given particle size and refractive index.

3.4 Color Composition

Once the scattering and refraction directions are determined for each wavelength, the light intensities are calculated. The phase function $P(\cos\theta)$ is used to model the angular distribution of scattered light:

$$P(\cos\theta) = \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}}$$

The scattered light intensities for red, green, and blue components are then computed:

$$I_{\text{scattered},\lambda} = P(\cos\theta) \cdot Q_{\text{sca},\lambda} \cdot (1 - Q_{\text{abs},\lambda})$$

where λ represents the wavelength (red, green, or blue).

The final color at each pixel is obtained by combining the scattered light intensities for the three wavelengths:

$$\text{Color}_{\text{final}} = (I_{\text{scattered, red}}, I_{\text{scattered, green}}, I_{\text{scattered, blue}})$$

3.5 Implementation Details

The `MieTransparentSphere` class handles the ray-particle intersection and the subsequent Mie scattering calculations. Here is a brief overview of the critical steps:

1. **Ray Intersection**: Determine if a ray intersects the particle using geometric calculations.
2. **Phase Function Calculation**: Compute the phase function for the intersection angle.
3. **Scattering Efficiency Calculation**: Precompute the scattering and absorption efficiencies for each wavelength.
4. **Refracted Directions**: Calculate the refracted directions for red, green, and blue wavelengths using Snell's law.

5. **Color Composition**: Combine the scattered light intensities to form the final color for the pixel.

The following snippet illustrates the computation of refracted directions:

```
double cosThetaT2Red = 1.0 - etaRed * etaRed * (1.0 - cosTheta * cosTheta);
if (cosThetaT2Red > 0) {
    Vector3 refractedDirectionRed = incomingDirection.clone().multiply(etaRed)
            .add(normal.clone().multiply(etaRed * cosTheta - Math.sqrt(cosThetaT2Red))).normalize();
    record.refractedRays[0] = new Ray(record.point, refractedDirectionRed);
}
```

The `generateSimulation` method orchestrates the entire process, from ray tracing to color composition, to produce realistic renderings of light interaction with nanoparticles. By integrating ray tracing with precise calculations of scattering and absorption efficiencies, the renderer produces realistic visualizations and, for me, some valuable insights into the optical properties of nanoparticles and their impact on light propagation.

4 Incorporating an Atmospheric Visualization

To enhance the visualization of Mie scattering, an "atmospheric" component is introduced around the particle. This atmosphere serves as a visual aid to emphasize the scattering pattern, allowing for a clearer understanding of how light interacts with the particle.

It is important to note that the effect of the white source light on this atmosphere (and the shadow that the sphere casts as a result) is excluded from the rendering process so as to isolate the effect of the scattered light.

4.1 Mathematical Modeling of the Atmosphere

The atmosphere is modeled by considering both scattering and absorption of light as it traverses through the medium. The key parameters in this model are the scattering coefficient σ_s and the absorption coefficient σ_a. These coefficients quantify the amount of scattering and absorption per unit length, respectively.

$$\sigma_s = 0.025, \quad \sigma_a = 0.1$$

The radiative transfer equation governs the intensity of light I as it propagates through the atmosphere:

$$\frac{dI}{ds} = -\sigma_a I + \sigma_s J$$

where J is the source term that accounts for the scattered light. For simplicity, the source term is approximated as a function of the light intensity and the scattering phase function $P(\cos\theta)$:

$$J = I \cdot P(\cos\theta)$$

The solution to this equation can be expressed as:

$$I(s) = I_0 e^{-\sigma_a s} + \sigma_s \int_0^s J(s') e^{-\sigma_a (s-s')} ds'$$

4.2 Implementation in the Rendering Process

The rendering process involves sampling the atmosphere along the path of each light ray. At each sampling point, the light contribution is computed and accumulated to form the final pixel color. The `Atmosphere` class encapsulates this functionality.

4.2.1 Ray Sampling in the Atmosphere

The light ray is sampled at regular intervals determined by the `samplingInterval` parameter. At each sample point, the light contribution is computed based on whether the light is blocked by the particle. If the light is not blocked, the contribution is added to the accumulated color:

```
public Color sampleAtmosphere(Ray ray, MieSphere sphere, DirectionalLight light, double maxDistance) {
    Color accumulatedColor = new Color(0.0, 0.0, 0.0);
    double distance = 0.0;

    while (distance < maxDistance) {
        Vector3 samplePoint = ray.at(distance);
        RayIntersect tempRecord = new RayIntersect();

        boolean blocked = sphere.hit(ray, tempRecord);

        if (!blocked) {
            Color lightContribution = new Color(0.0, 0.0, 0.0);
            Ray lightRay = new Ray(samplePoint, light.direction);
            RayIntersect lightIntersect = new RayIntersect();
            if (!sphere.hit(lightRay, lightIntersect)) {
                // lightContribution = lightContribution.add(light.color);
            }

            Color surfaceLight = sphere.getSurfaceLight(samplePoint);
            lightContribution.add(surfaceLight);

            accumulatedColor.add(lightContribution.multiply(scatteringCoefficient));
        } else {
            break;
        }

        distance += samplingInterval;
    }

    return accumulatedColor.multiply(Math.exp(-absorptionCoefficient * distance));
}
```

4.2.2 Combining Atmospheric and Particle Contributions

For each pixel, the renderer determines if the ray intersects the particle. If an intersection occurs, the particle's scattering properties are used to compute the pixel color. Otherwise, the atmospheric contribution is computed and added to the pixel color.

```
for (int y = 0; y < imageHeight; y++) {
    for (int x = 0; x < imageWidth; x++) {
        double u = (double) x / (imageWidth - 1);
        double v = (double) y / (imageHeight - 1);
        Ray ray = camera.getRay(u, v);

        RayIntersect record = new RayIntersect();
        if (mieSphere.hit(ray, record)) {
            Color pixelColor = record.modifiedColor;
            image.setRGB(x, imageHeight - 1 - y, pixelColor.toInt());
        } else {
            Color atmosphereColor = atmosphere.sampleAtmosphere(ray, mieSphere, light, atmosphereDepth);
            image.setRGB(x, imageHeight - 1 - y, atmosphereColor.toInt());
        }
    }
}
```

4.3 Visualization Results

The addition of the atmospheric component enhances the visual representation of Mie scattering patterns. The gradual accumulation of light contributions as rays traverse the atmosphere provides a more vivid depiction of the scattering phenomena and the spatial distribution of the scattered light.

The addition of an atmospheric component in the rendering process significantly improves the visualization of Mie scattering effects. By accurately modeling the scattering and absorption of light within the atmosphere, the renderer provides a comprehensive and realistic depiction of light behavior around nanoparticles.

As can be seen from the following example renders, the process is extremely sensitive to minute differences in variables such as size and refractive index, often leading to very interesting and visually appealing results.

5 Example Renders

The following is a series of images rendered from a fixed point of view with the camera facing the center of the particle, in this case a water droplet with a refractive index of 1.33. The RGB white light source is located out of view toward the top left of the images, and radiates parallel light rays toward the bottom right.

By varying the size of the particle, many beautiful and interesting scattering patterns become apparent.

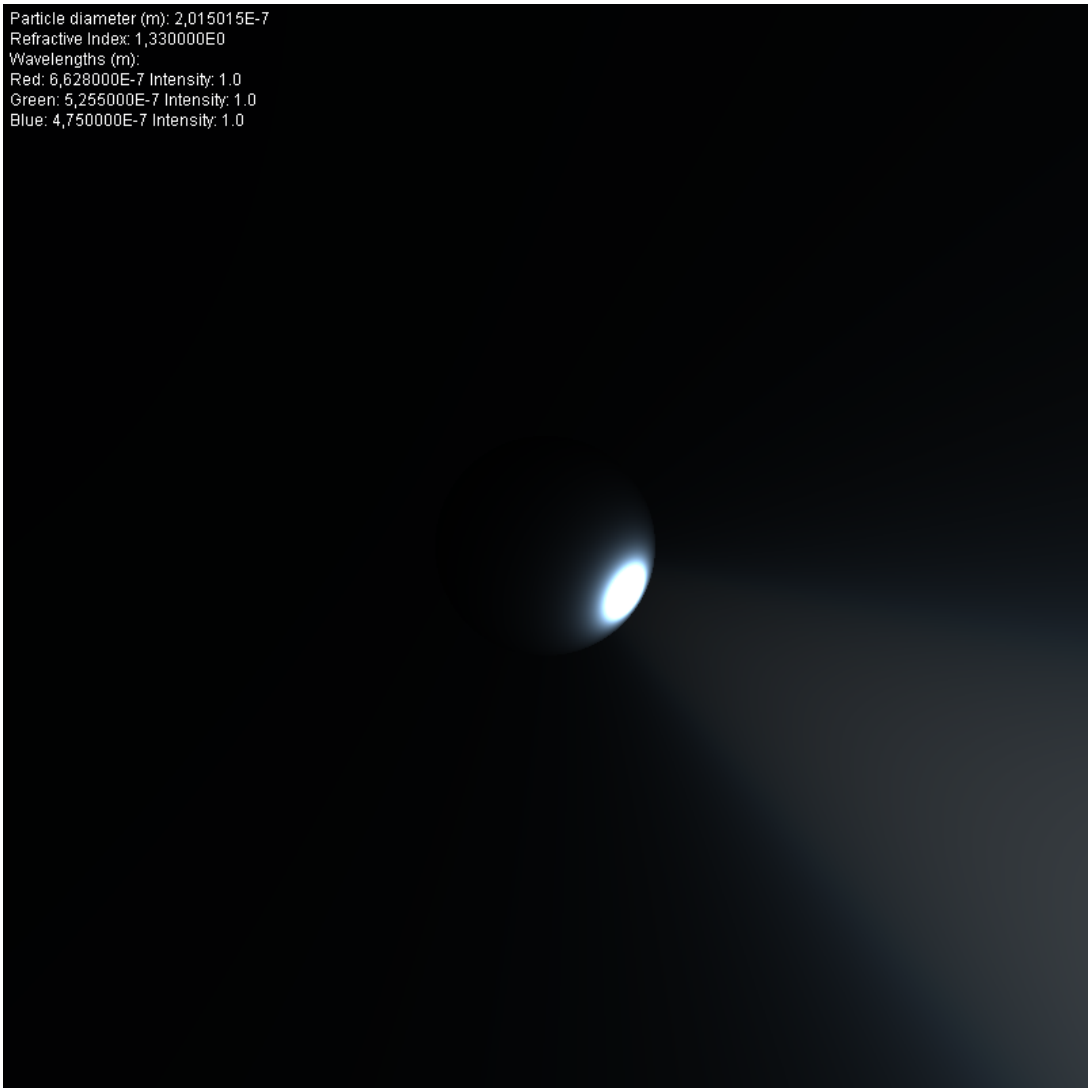

Figure 1: Particle diameter: 201.5015 nm *The white light is emitted in a narrow forward beam.*

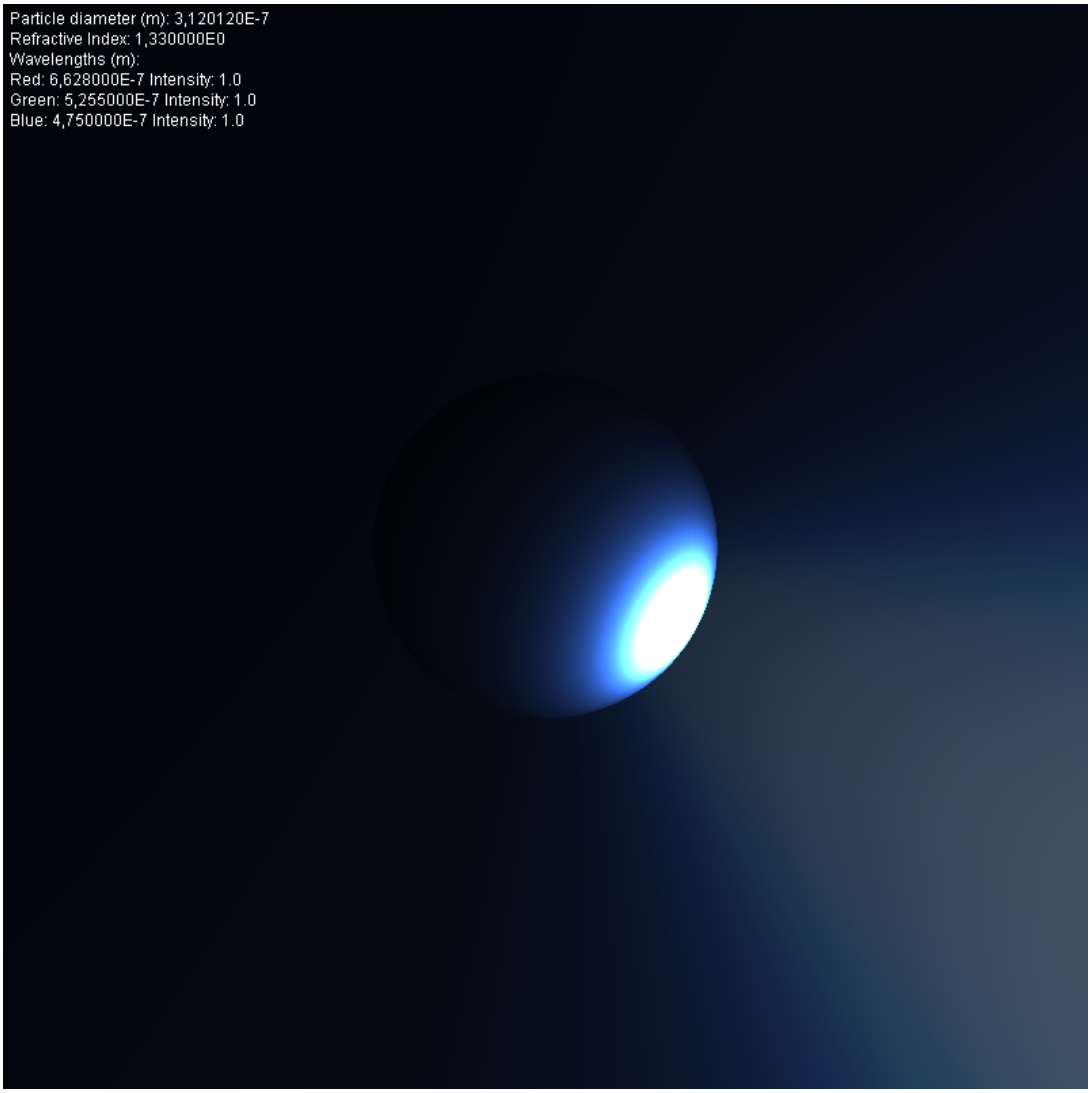

Figure 2: Particle diameter: 312.012 nm *The light is mostly scattered forward in a narrow beam, but the blue fringe indicates a greater amount of blue light scattering.*

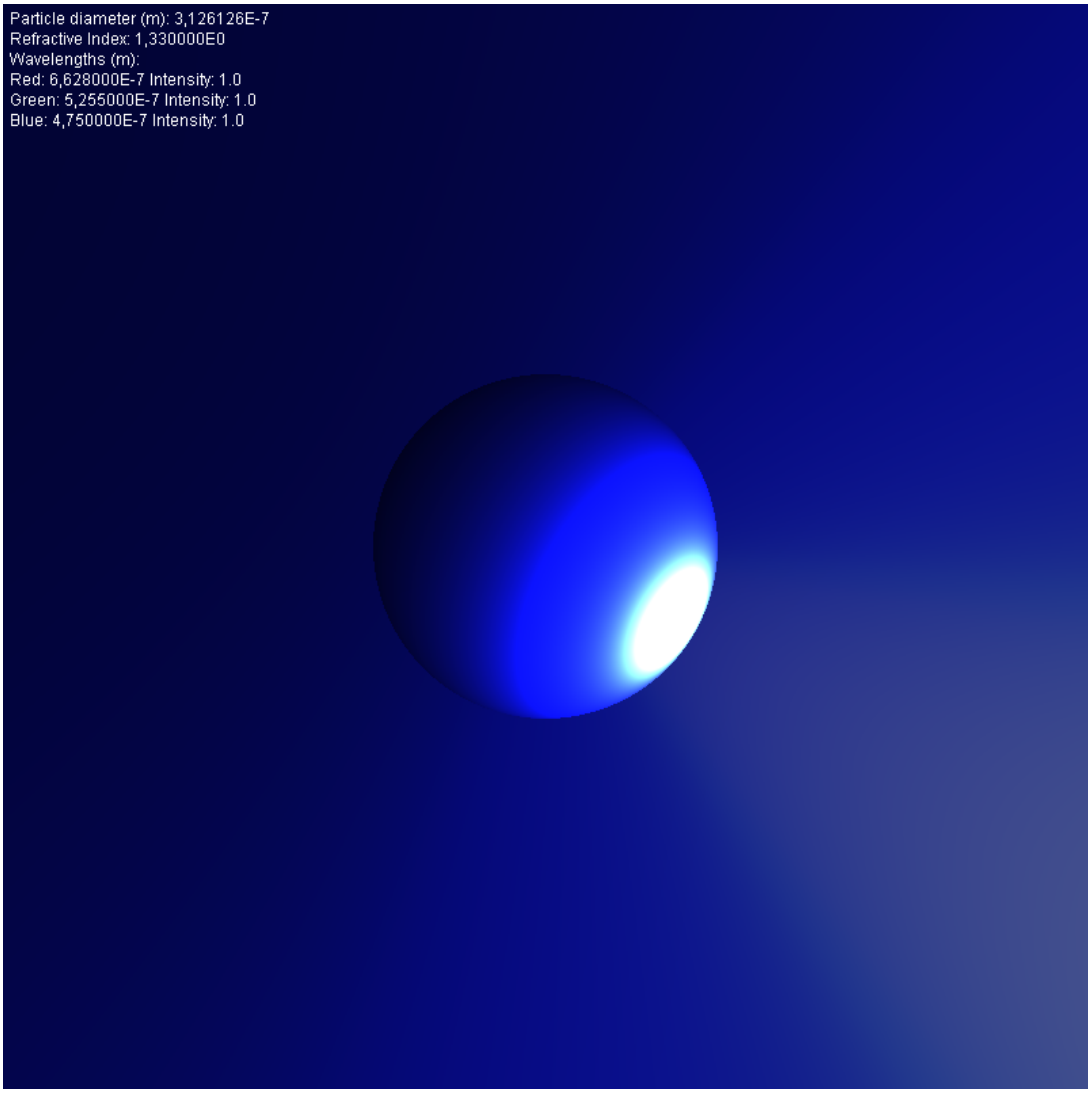

Figure 3: Particle diameter: 312.6126 nm *The light is mostly scattered forward in a narrow beam, but the blue band indicates a strong scattering of blue light.*

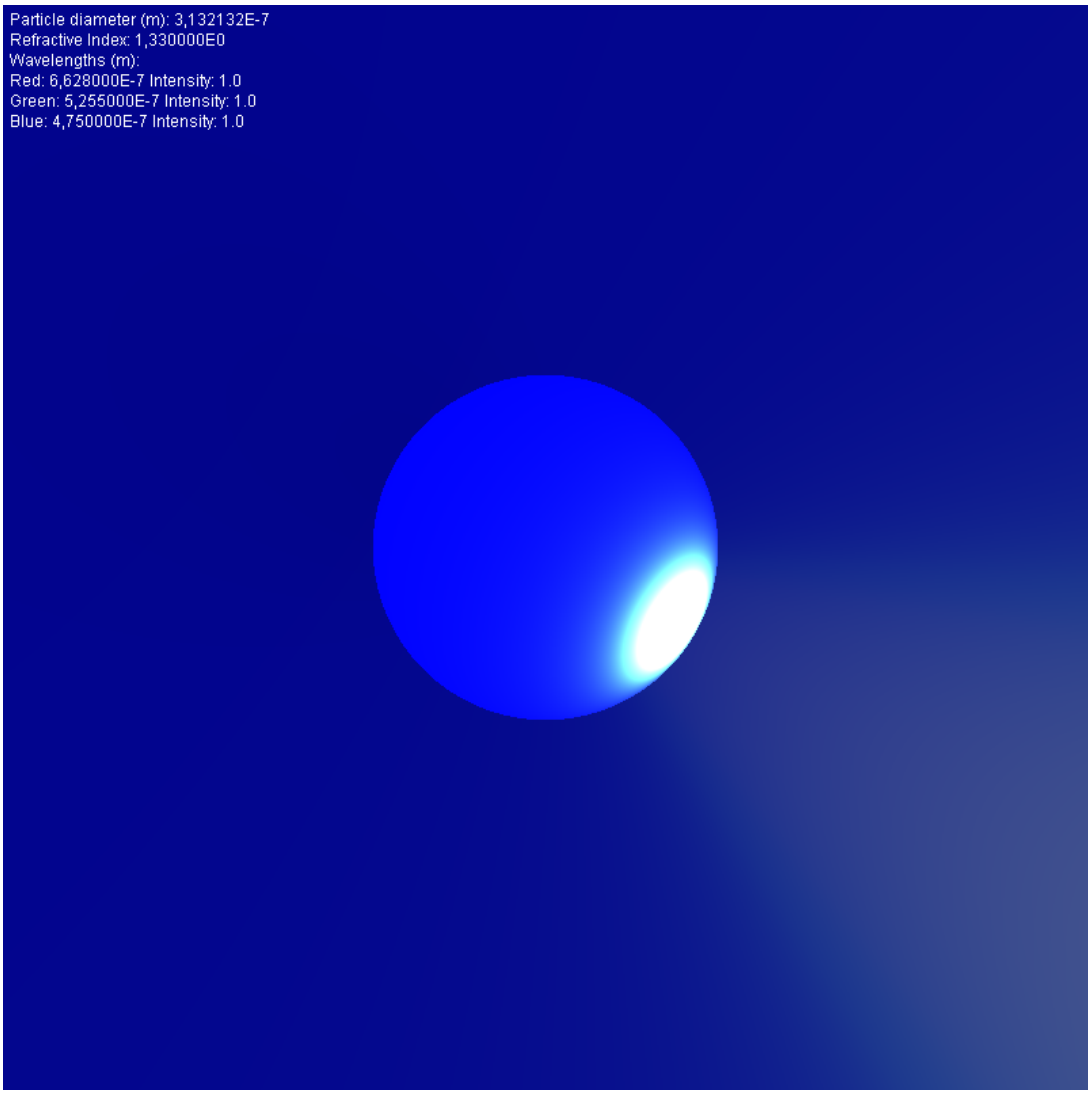

Figure 4: Particle diameter: 313.2132 nm *The red and green components of the white light are scattered forward in a narrow beam, but blue light is scattered almost evenly in all directions, giving the particle a self-illuminating appearance.*

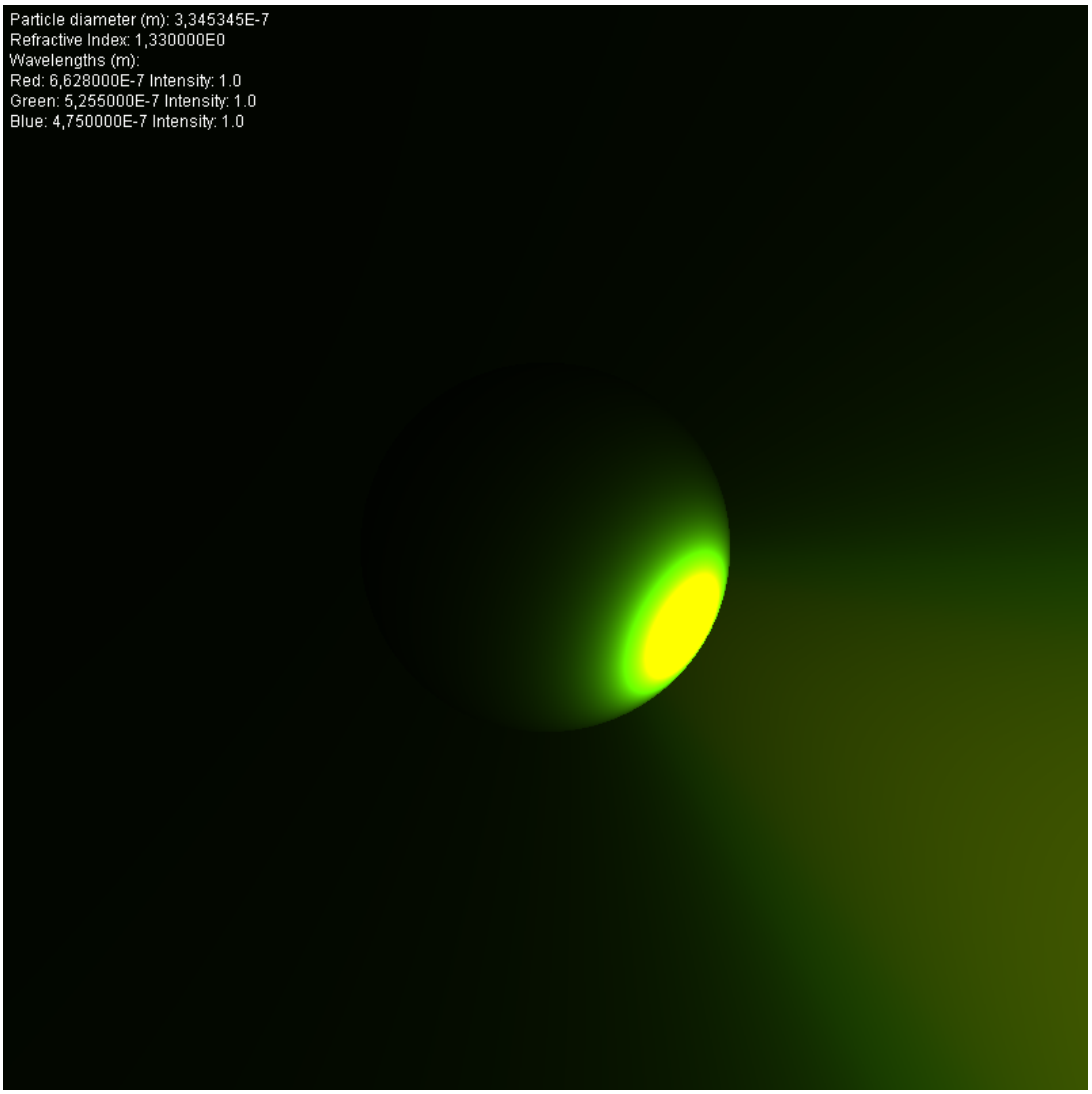

Figure 5: Particle diameter: 334.5345 nm *The blue light is almost entirely internally reflected, and only the green and red components contribute to the scattered light. The green light is scattered more than the red light, giving off a yellow-green glow.*

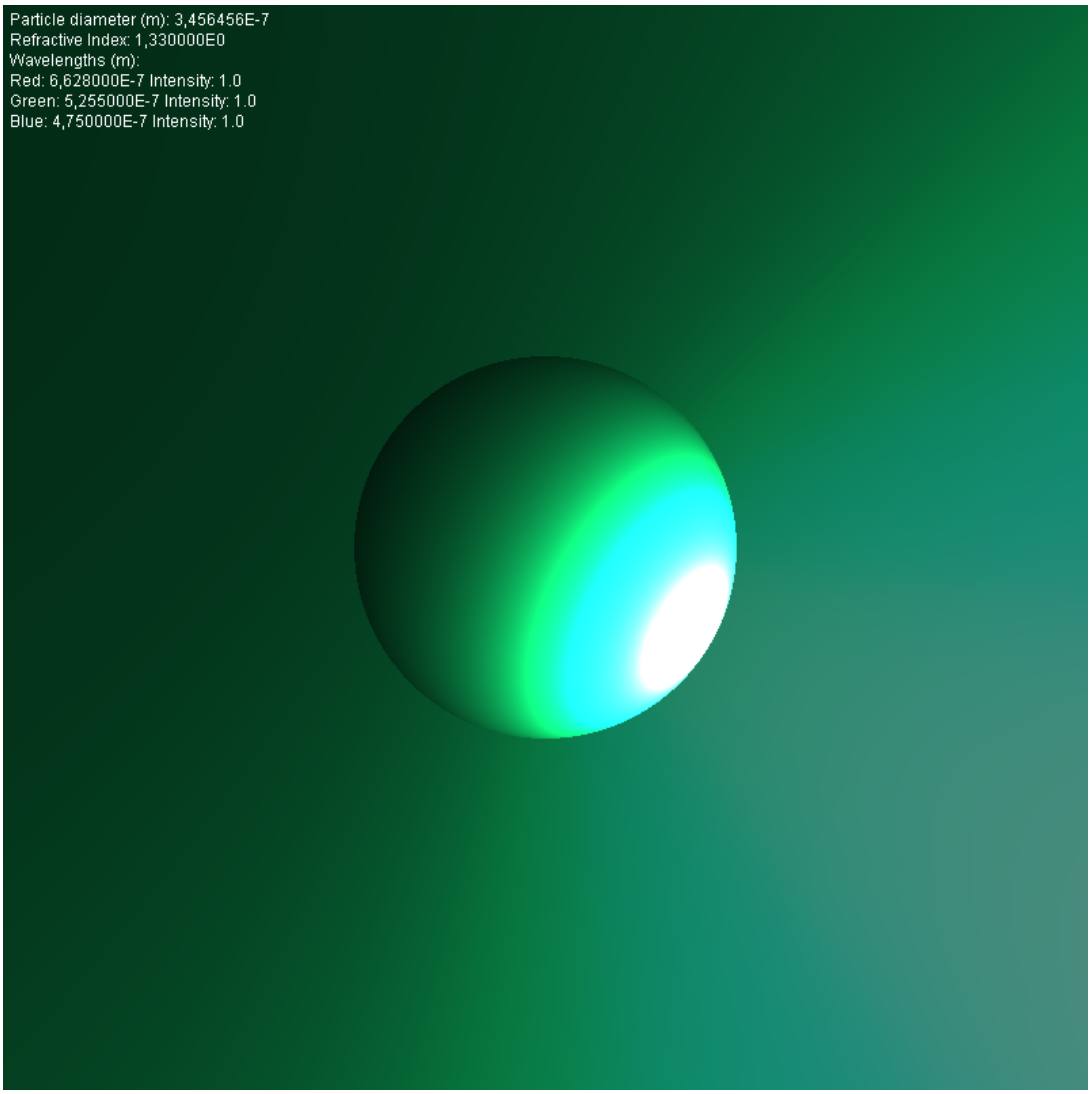

Figure 6: Particle diameter: 345.6456 nm *Despite having a shorter wavelength than green light, here the blue light is scattered less than the green light, resulting in a out-of-order rainbow effect and a distinct turquoise glow.*

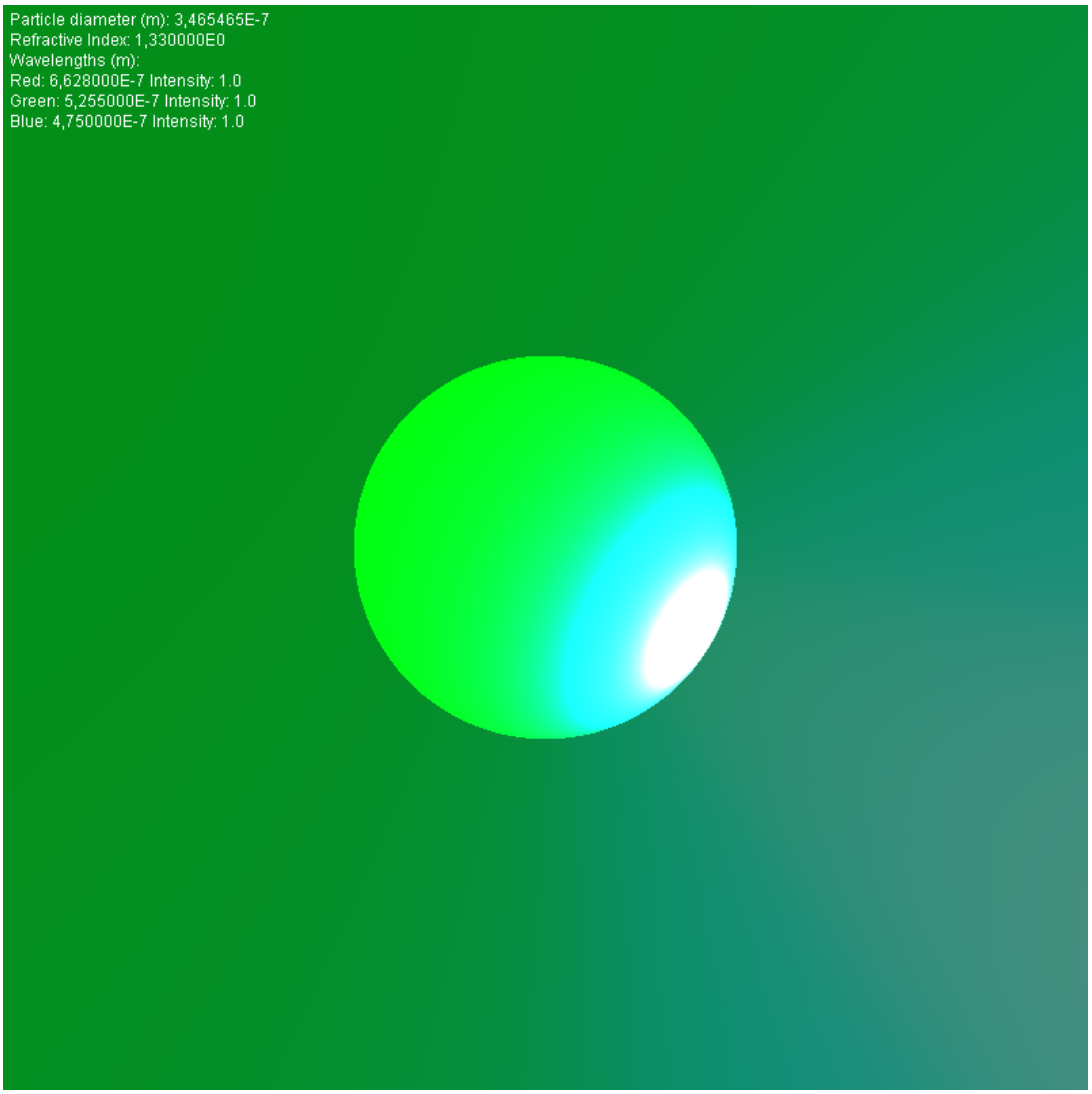

Figure 7: Particle diameter: 346.5465 nm *The red and blue components of the white light are scattered forward in a narrow beam, but green light is scattered almost evenly in all directions, giving the particle a self-illuminating appearance.*

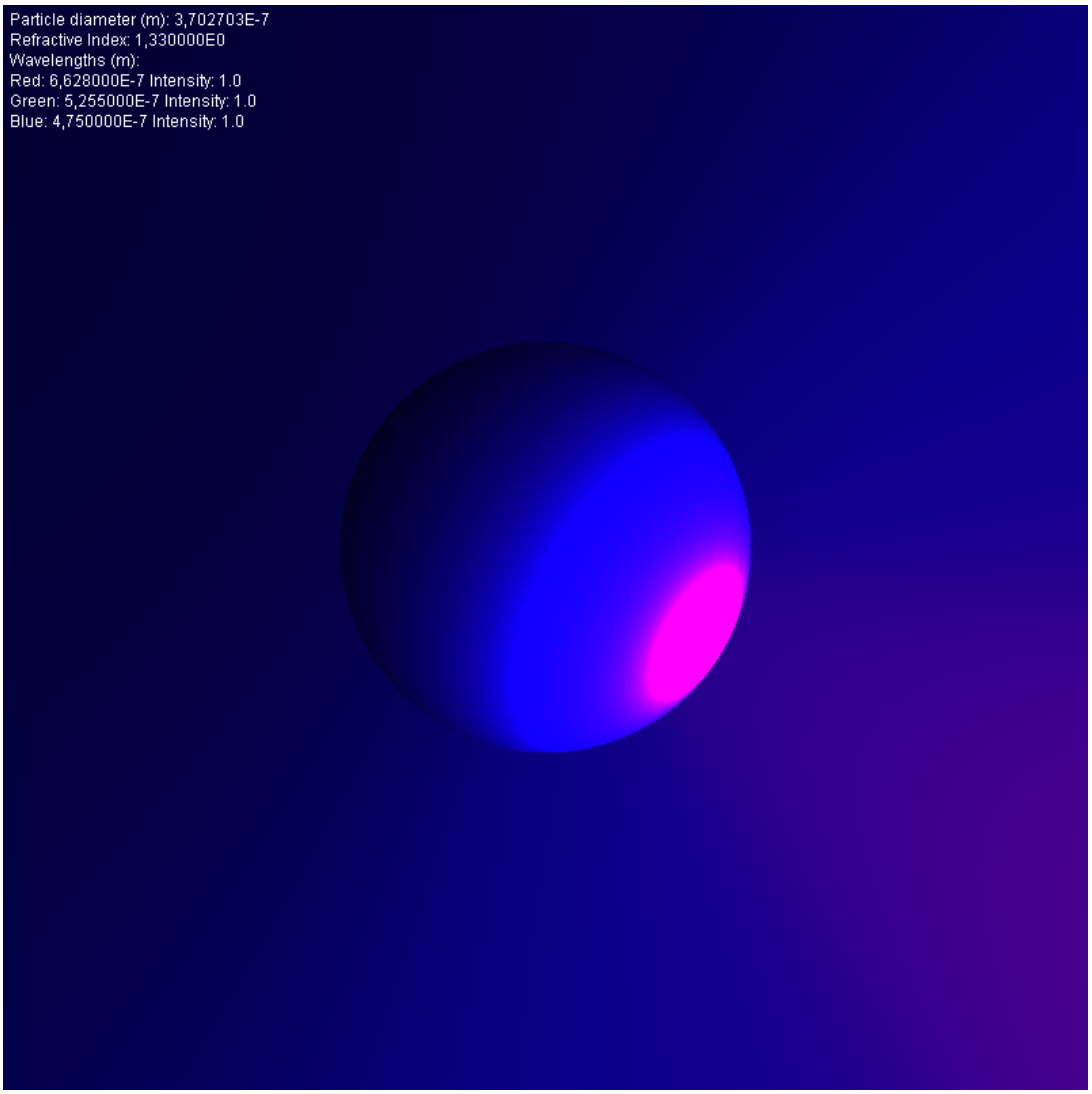

Figure 8: Particle diameter: 370.027 nm The green light is almost entirely internally reflected, and only the blue and red components contribute to the scattered light. The blue light is scattered more than the red light, giving off a violet-blue glow.

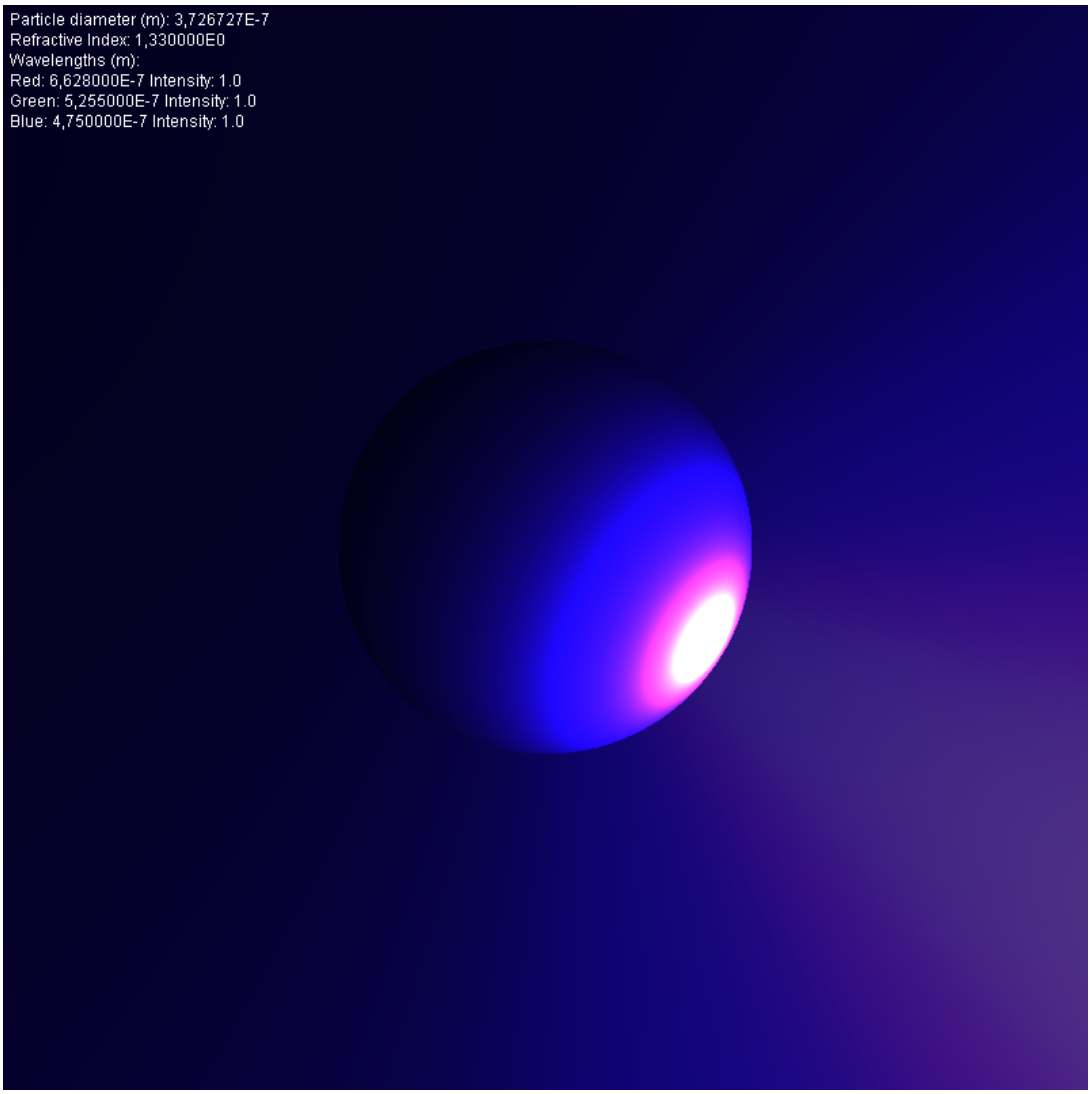

Figure 9: Particle diameter: 3472.6727 nm *Despite having a shorter wavelength than red light, here the green light is scattered less than the red light, resulting in a out-of-order rainbow effect and an interesting scattering pattern.*

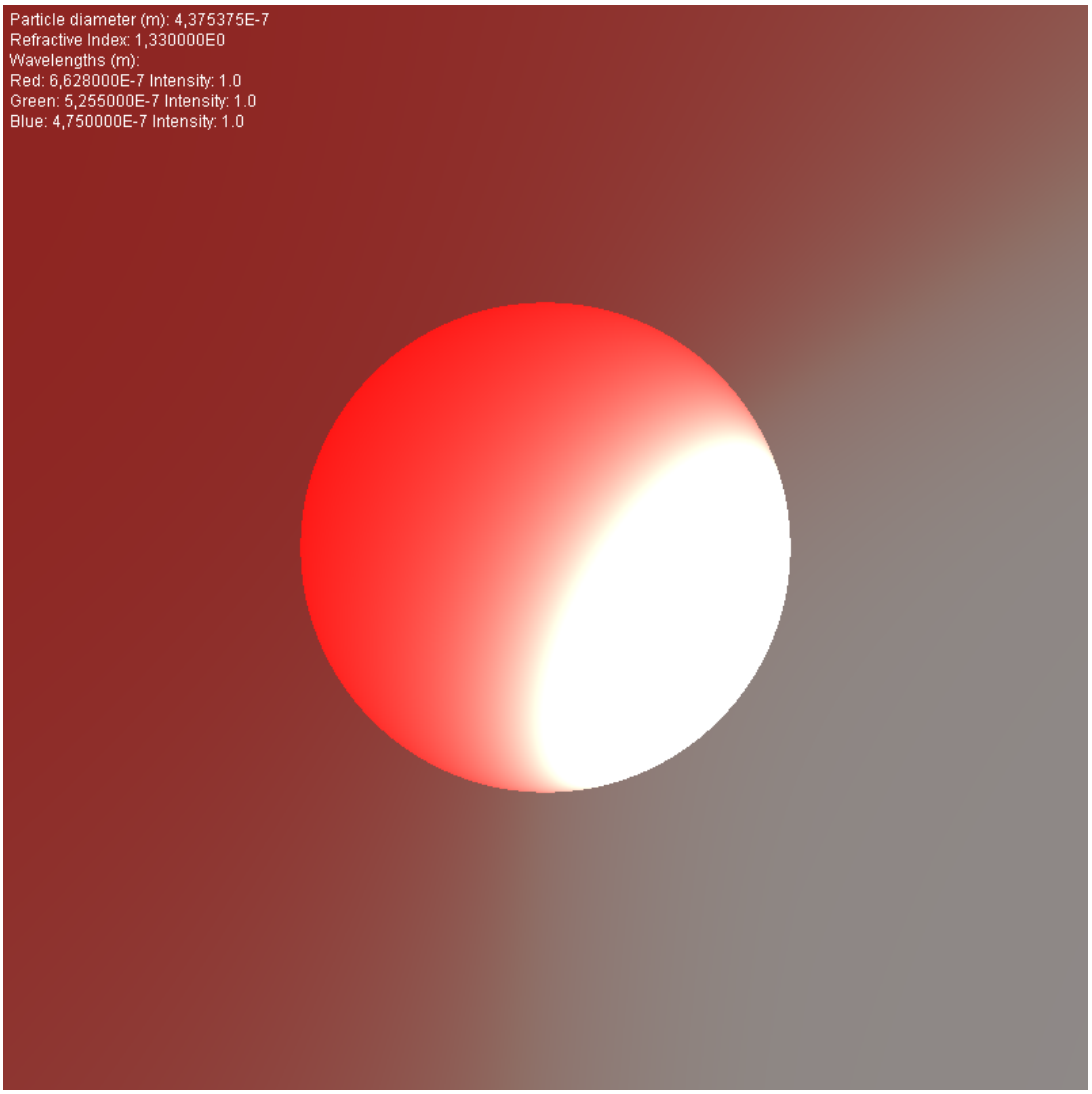

Figure 10: Particle diameter: 437.5375 nm *The blue and green components of the white light are scattered forward in a broad beam, but red light is scattered almost evenly in all directions, giving the particle a self-illuminating appearance.*