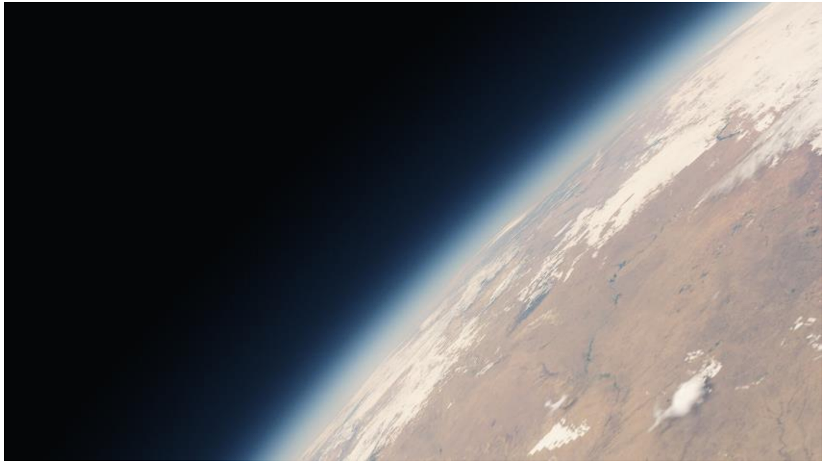



Defferred Atmospheric Scattering

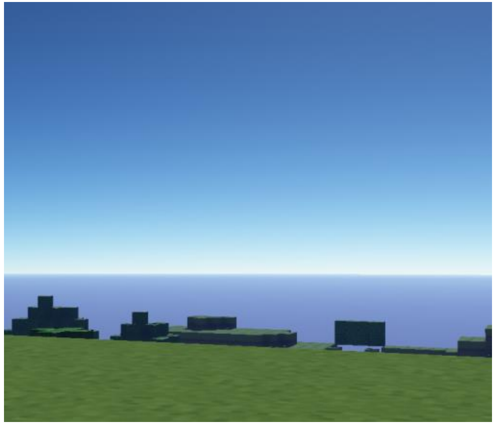
Iñigo Fernández Arenas

CS562

What is Atmospheric Scattering?

Atmospheric Scattering occurs when a particle in the atmosphere **changes the direction of light ray.**

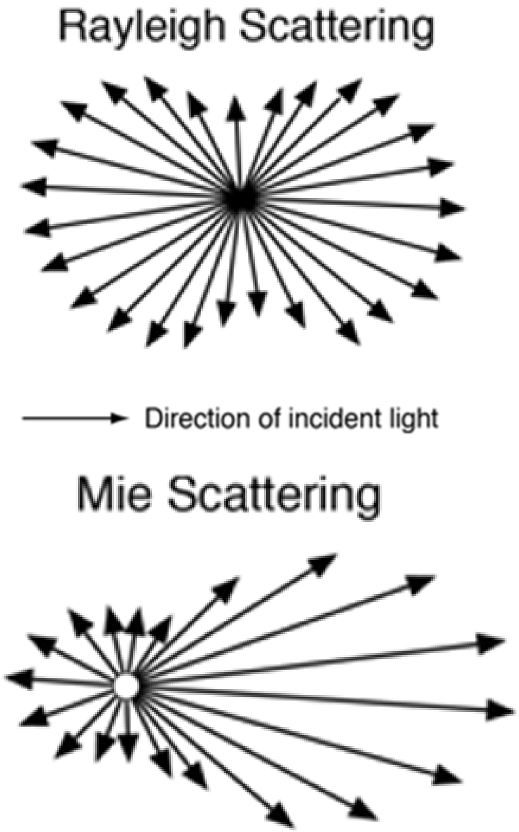
Responsible for **bluish sky** when the sun is above the ground or the reddish sky while sunsets. It is also responsible for the **halo** created outside the planet.



Types of Scattering

Rayleigh Scattering

- ✓ Usually represents small particles in the air.
- ✓ Favors short wavelengths, therefore blue colors are scattered more than other colors.
- ✓ Produces the color to be blue or red.



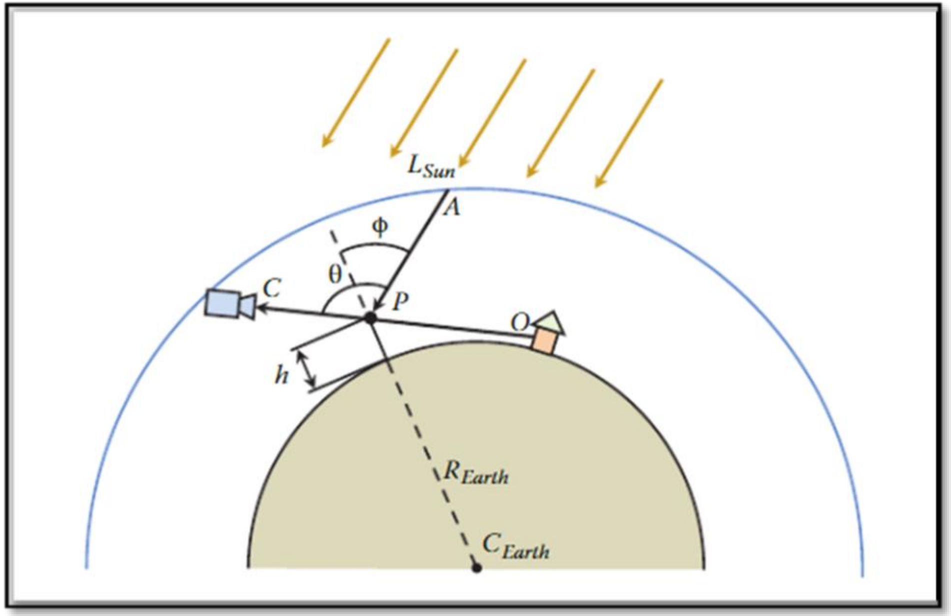
Mie Scattering

- ✓ Represents aerosols in the or other greater particles in the air.
- ✓ It is almost not wavelength dependent.
- ✓ It can produce fog, glare around the sun and other effects.

Physical Theory

Phase functions describe the angular distribution of scattered light for a specific wavelength.

$$p_R(\emptyset) = \frac{3}{16\pi} * (1 + cos^2(\emptyset))$$



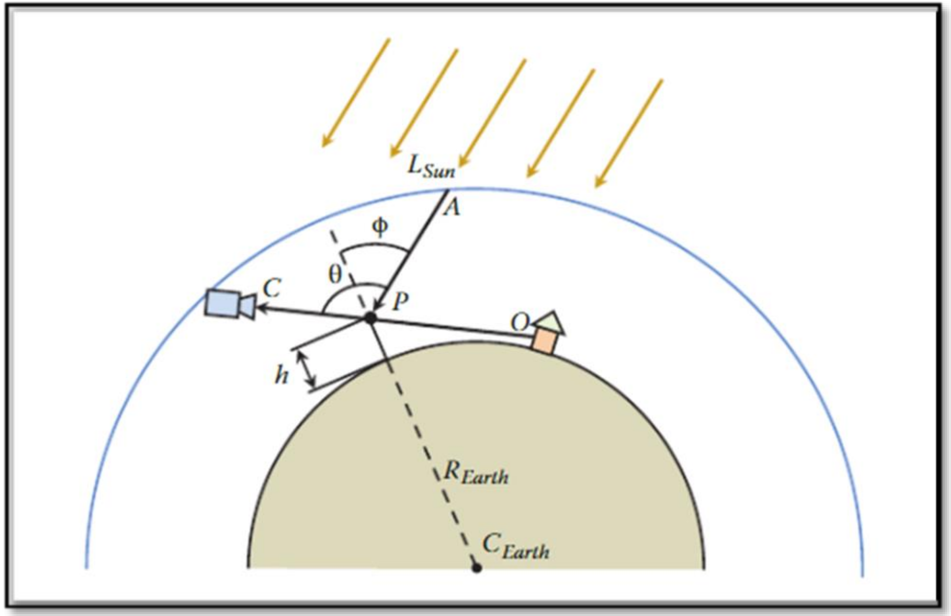
$$p_M(\emptyset) = \frac{1}{4\pi} * \frac{3 * (1 - g^2)}{2 * (2 + g^2)} * \frac{(1 + cos^2(\emptyset))}{(1 + g^2 \ - 2g * cos(\emptyset))^{3/2}}$$

- p_R = Rayleigh phase function.
- p_M = Mie phase function.
- g = constant value that depends on implementation.
- ∅ = Angle between sun direction and camera direction.

Physical Theory

Optical Depth along the path A to B is the integral of the total extintion coefficient and is given by:

$$T(A \to B) = \int_A^B \left(B_R^e * e^{-\frac{h(s)}{H_R}} + B_M^e * e^{-\frac{h(s)}{H_M}} \right) * dt$$



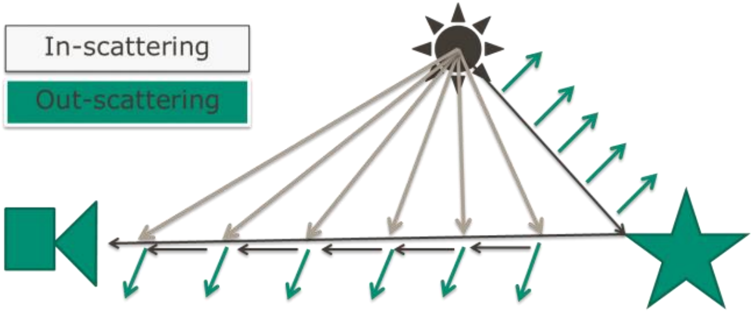
- B_R^e = Rayleigh extinction coefficient, constant in the implementation.
- B_M^e = Rayleigh extinction coefficient, constant in the implementation.
- H_R = Rayleigh Scale Height, constant in the implementation.
- H_M = Mie Scale Height, constant in the implementation.

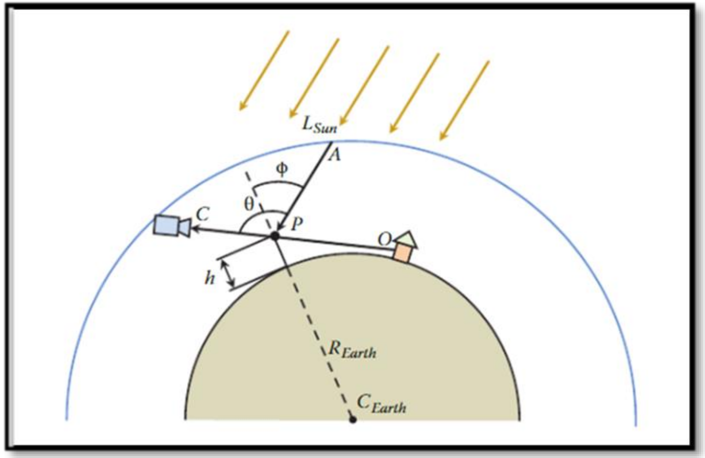
Physical Theory

- The in-scattering light represents the amount of light that is redirected from a particle towards the camera:

$$L_{in} = \int_{C}^{O} L_{sun} * e^{-T(A(s)\rightarrow P(s))} * e^{-T(P(s)\rightarrow C)} * V\big(P(s)\big) * \\ * \left(B_R^S * e^{-\frac{h(s)}{H_R}} * p_R(\emptyset) + B_M^S * e^{-\frac{h(s)}{H_M}} * p_M(\emptyset)\right) ds$$

- L_{sun} = Light of the sun.
- $P(s)$ = Current point in the integrated line.
- $A(s)$ = Intersection point in the atmosphere with the sun direction towards the point $P(s)$.
- $V(P(s))$= Value that determines if $P(s)$ is in sadow.





- The total light is computed by the sum of the in-scattered light ad the reflected light from a Surface attenuated by the path:

$$L = L_o * e^{-T(O\rightarrow C)} + L_{in}$$

- L_o = Light reflected from the surface sampled.

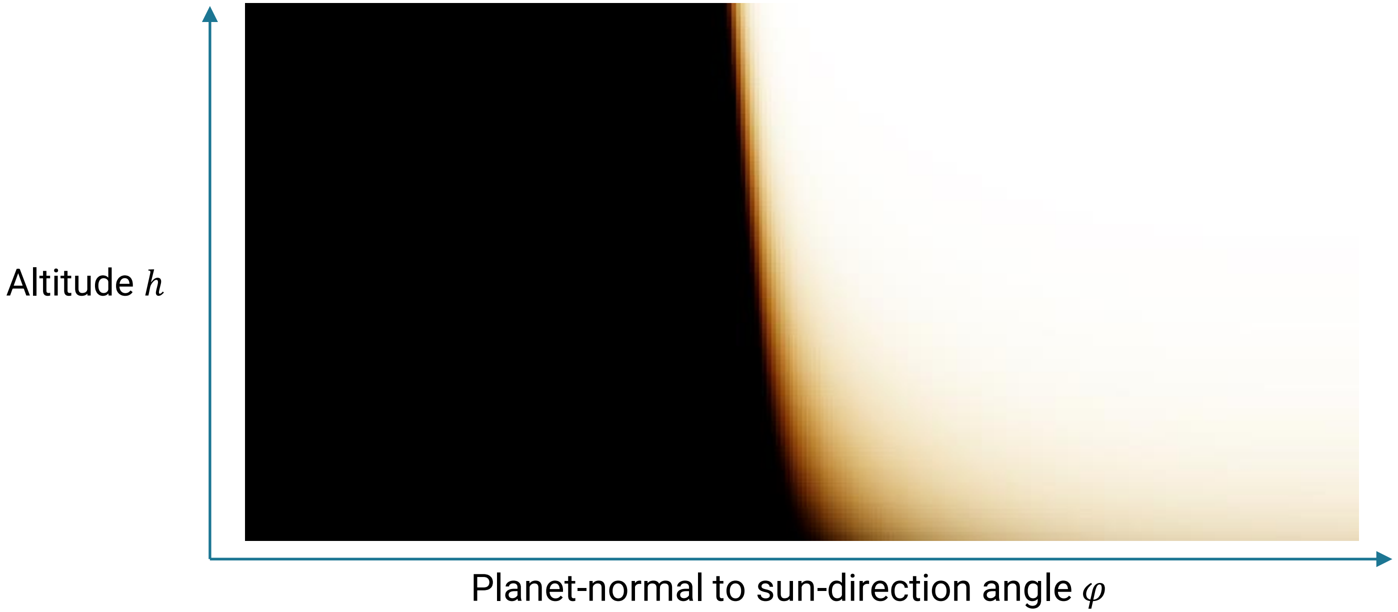
Implementation

- Remove two internal integrals in order to solve it in real time:
  - $T(P(s) \rightarrow C)$: since it is evaluating the same path as the outer integral it can be computed at the same time updating the total density for each sample of the numeric integration.

```
for(int s = 0; s <= inscatter_steps; s++)
{
    vec3 p = origin + dir * float(s);
    vec3 normal_to_planet = p - planet_center;
    float alt = length(p - planet_center);

    vec2 deltaRM = vec2(exp(-(alt - planet_radius) / HR), exp(-(alt - planet_radius) / HM));
    density_CP += deltaRM * ds;
    vec3 Ecp = exp(-(density_CP.x * Br + density_CP.y * Bme));

    vec3 Eap = GetExtintionToAtm(alt, normalize(normal_to_planet));
    vec3 Eapc = Eap * Ecp;

    vec4 view_pos = inverse(ViewToWorld) * vec4(p, 1.0f);
    float v = 1.0f;
    if(light_shafts && ComputeShadow(view_pos.rgb) != 0.0f)
        v = 0.0f;

    Lrigh += deltaRM.x * Br * Eapc * v * ds;
    Lmie  += deltaRM.y * Bms * Eapc * v * ds;
}
```

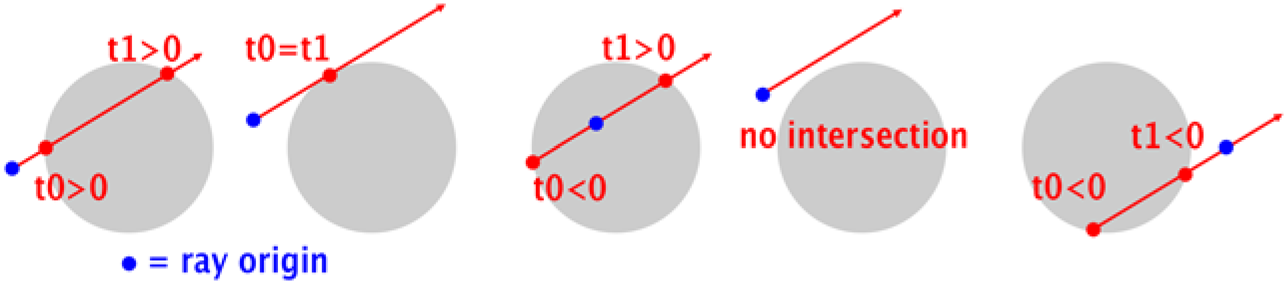
Implementation

- $\boldsymbol{T}(\boldsymbol{A}(\boldsymbol{s}) \rightarrow \boldsymbol{P}(\boldsymbol{s}))$: Since this integral only depends on the altitude h and an angle φ between the normal to the surface and the sun direction, we can precompute a lookup table storing this information. In my case I precomputed a three-channel lookup table storing the result of $e^{-T(A(s)\rightarrow P(s))}$ giving me a result similar to this texture.



Implementation

- It is almost mandatory to **compute** $\boldsymbol{V(P(s))}$ using **cascaded shadow maps**, considering the magnitude of the scene.
- Finally, we need to compute the **end points of the raymarching integration** $\boldsymbol{O \to C}$**.** For this we need a **ray vs sphere intersection test**, and we can have different scenarios:
  - **Camera Inside Atmosphere**
    - O: the camera itself
    - C: it can be a surface point or the top of the atmosphere
  - **Camera Outside Atmosphere**
    - O: the entry point of the view ray with the atmosphere
    - C: it can be a surface point of the planet or the exit point of the view ray with the atmosphere
    - Extra: the view ray may not intersect the planet therefore no integration will be computed



- Once we have everything previously explained clear we can **perform the raymarching and simulate an atmosphere.**

Conclusions

Using a small amount of **integration steps** (20-50) gives a **good in-scattering result** with **a good performance time**, but the **light shafts** created by the shadows **suffer** a lot in precision because of the small number of steps.

↓

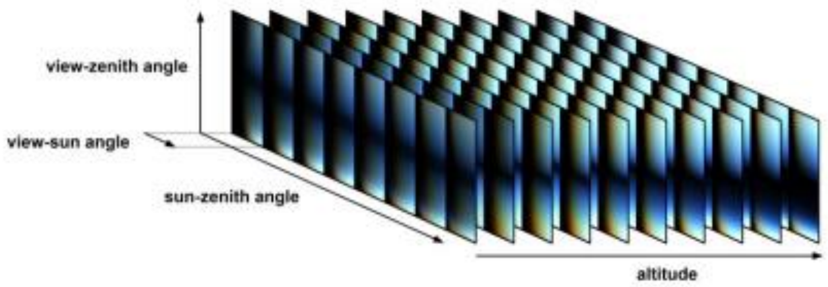
So, we should think of **a way to handle this effect separately** in order to obtain a better result in real time.

20 steps, real time

800 steps, not real time

Conclusions

- There are **other aproaches** that precompute also the in-scattering value for every direction posible, we would need to use a 2D_ARRAY_TEXTURE or a 3D_TEXTURE, and in this case, we must treat shadows separately.
  - We would **gain performance**, but the amount of **memory** contained in that texture loaded into the graphics card will **grow a lot**.
  - There will be a **new artifacts to consider**.
  - But I think is a **worth try** if we want to implement this effect into a game.



Conclusions

- We could implement more effects to create a more realistic result with a **lens flare** post process effect or **simulate clouds** in the atmosphere.

Demo Time