RENDERING ATMOSPHERIC LIGHT SCATTERING

V1.0 Single Scattering

TFG - Gustavo Raush Faggembauu

LaSalle URL

2017-2018

OUTLINE

INTRODUCTION

BASICS

SCATERRING THEORY

CAVEATS OF NAIVE IMPLEMENTATION

ACTUAL IMPLEMENTATION

DEMO

CONCLUSIONS AND FUTURE WORK

REFERENCES

OUTLINE

Who am I?

CL3VER

- Multimedia Engineering student
- Intership at Cl3ver, *1 year and 5 months*
- TFG at atmospheric light scattering
- Interested in Computer Graphics (mostly real time)

OUTLINE

Introduction – Atmospheric Light Scattering

- Difficult problem but very important for outdoor environments rendering
- Difficult complex equations, hard to achieve real time
- Real-Time algorithm running on GPU using methods described by Nishita et al. 1993. [REF1]
- Paper done by Sean O'Neil, described at Nvidia GPU Gems2 [REF2]

Atmospheric Scattering

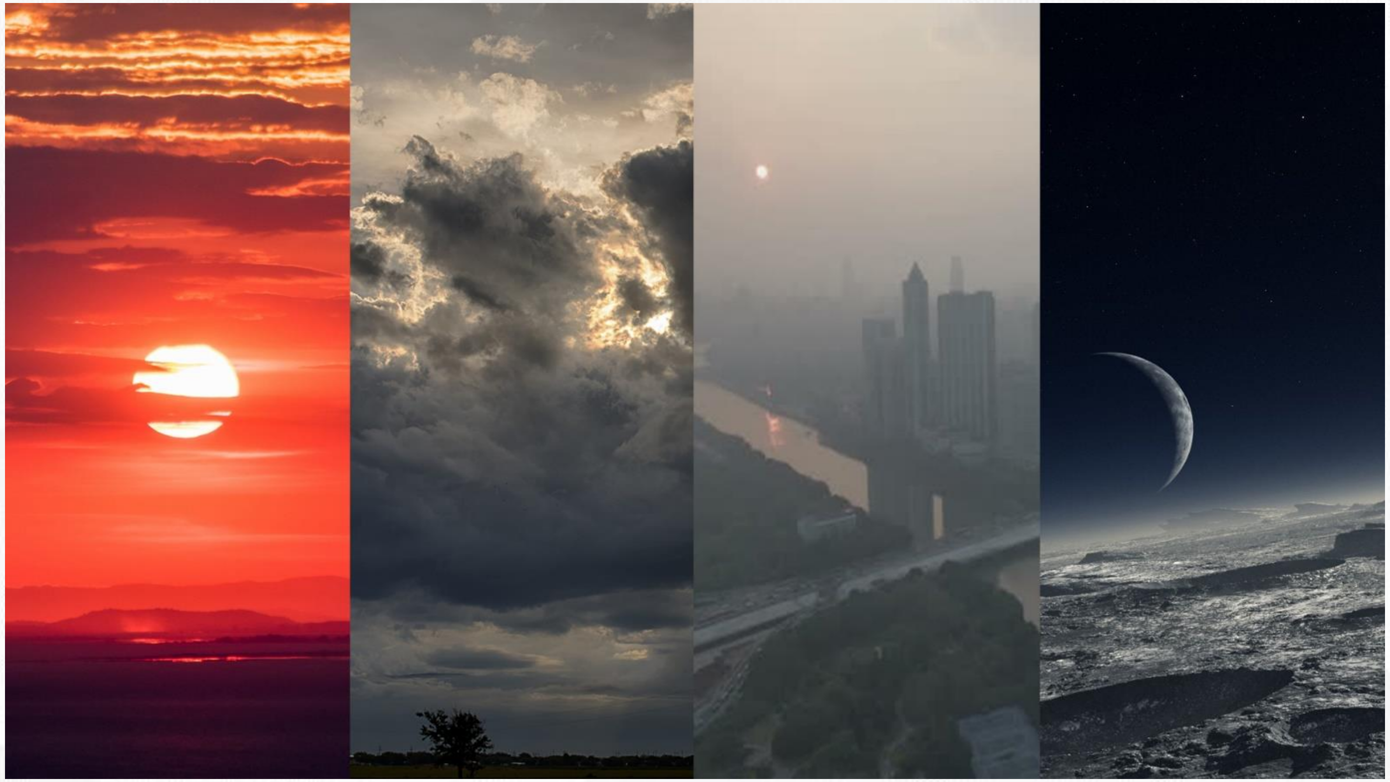
- Sky color
- Fog
- Clouds
- “God rays”
- Light shafts
- Volumetric shadows

Atmospheric Light Scattering

- Atmosphere scatters light
- Responsible for the sky color
- Caused by a variery of particles
  - Molecules, dust, water vapor, etc.
- Illuminates the sky
- Attenuates and colors the Sun
- Attenuates and colors distant objects (aerial perspective)

Atmospheric Light Scattering

- Varies by
  - Time of day
  - Weather
  - Pollution
  - Planet atmosphere



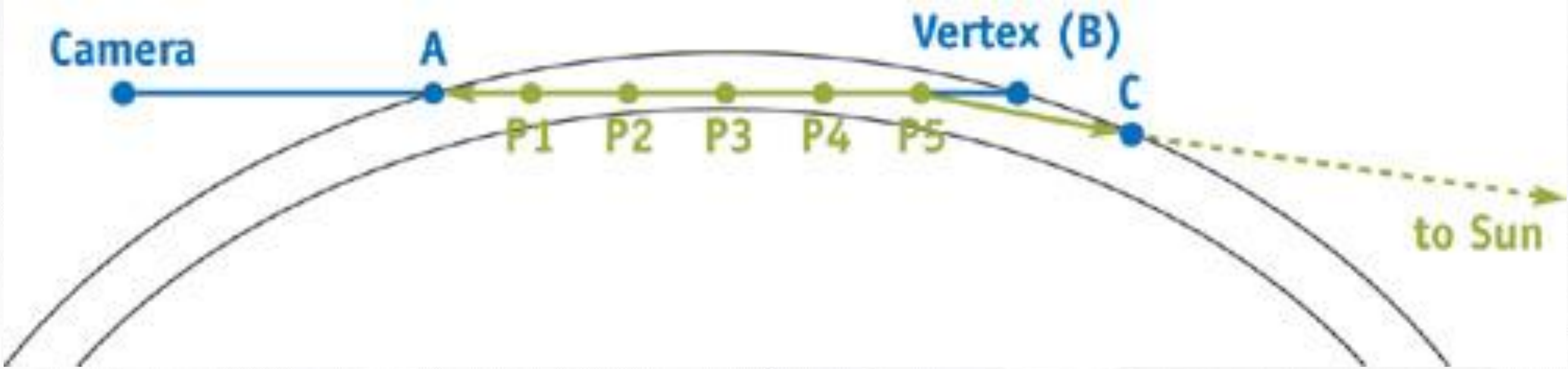
How we will compute it? (1)

- We have a ray from the camera through the atmosphere to a vertex
- Vertex can be part of terrain, part of sky dome, part of cloud or object in space
- If ray passes through atmosphere scattering needs to be calculated
- A ray has two points defined, where starts passing through atmosphere and where it stops. A and B respectively
- Depends on the situation we have to do a sphere-check intersection to find the point. (camera in space; or camera in atmosphere)

How we will compute it? (2)

- We have a line segment defined from point A to B
- Our goal is to approximate the integral that describes atmospheric scattering across its length
- **Lets take five sample positions, $\boldsymbol{P_1}$ through $\boldsymbol{P_5}$**. Each point represents a point in the atmosphere where light scatters
- At P_5 light comes from the sun, in a straight line. At P_5 some of this light is scattered directly towards the camera.
- As this light travels to the camera, some of it gets scattered away again
- The same happens for every sample point
- For calculating the integral: The **midpoint rule method**

How we will compute it? (3)



The midpoint rule

-Method for approximating a definite integral
-Weighted sum calculated in a loop

MIDPOINT RULE:

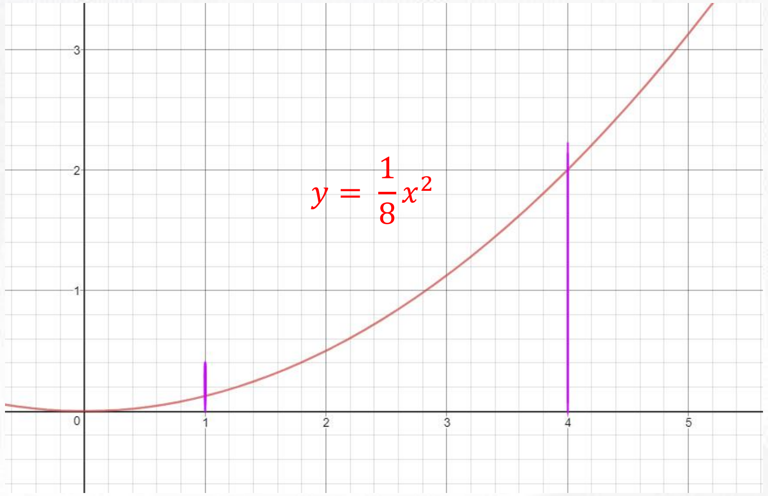
- Definite integral
- Break definite integral into n segments
- Evaluate integral at center of each segment
- Multiply each result by length of segment
- Add them up

-PROFIT!

The midpoint rule

-EXAMPLE

Use the Midpoint rule with n = 4 to approximate the definite integral $\int_1^4 \frac{1}{8}x^2\,dx$



The Midpoint Rule

$$Area \approx \sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right)\Delta x \qquad \Delta x = \frac{b-a}{n} \qquad x_i = \mathrm{a} + \mathrm{i}\Delta x$$

$n = 4$
a = 1
b = 4

$$\Delta x = \frac{4-1}{4} = \frac{3}{4} \qquad x_0 = 1 \qquad x_1 = 1 + \frac{3}{4} = \frac{7}{4} \qquad x_2 = 1 + 2\frac{3}{4} = \frac{10}{4}$$

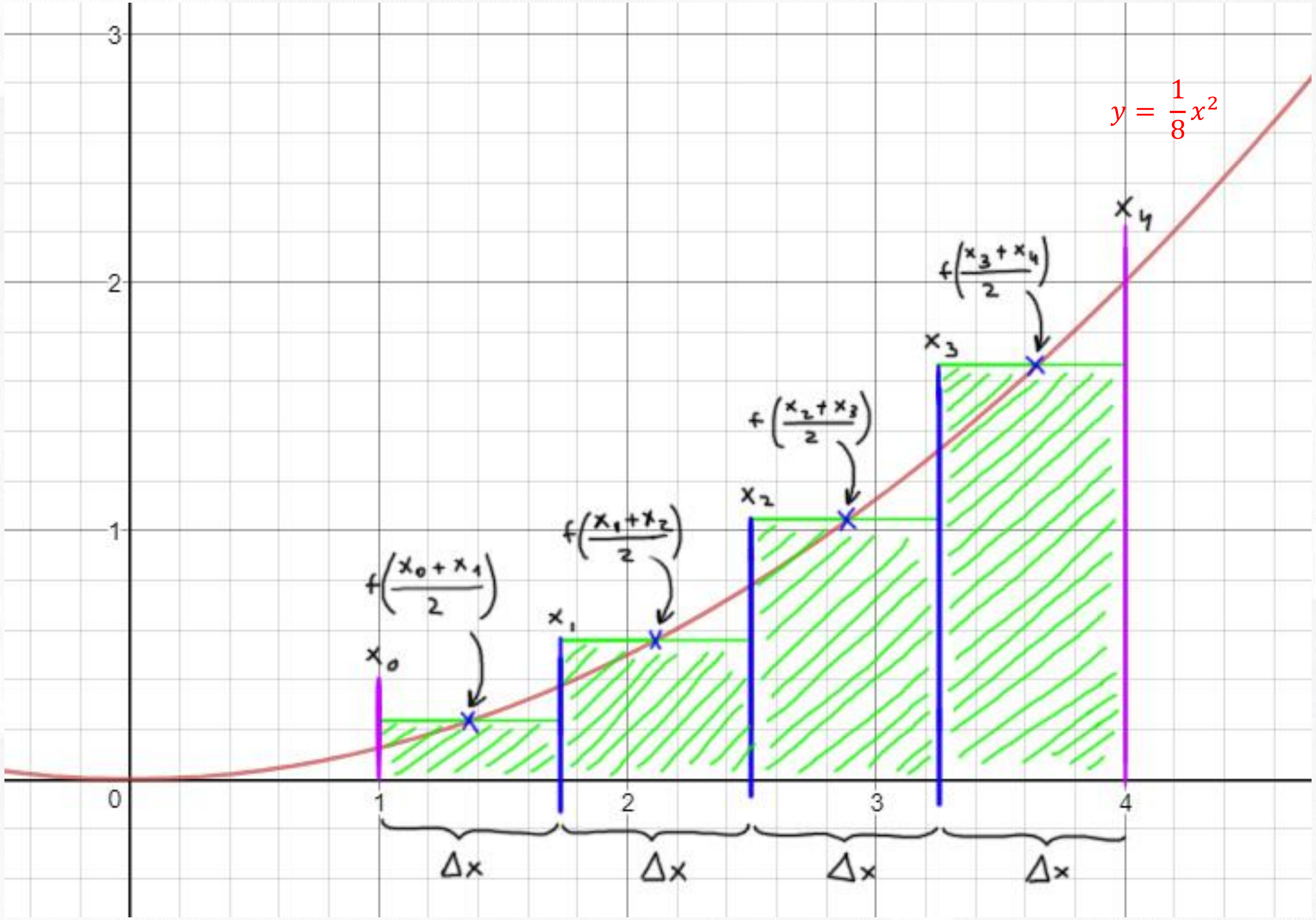
$$x_3 = 1 + 3\frac{3}{4} = \frac{13}{4} \qquad x_4 = 1 + 4\frac{3}{4} = 4$$

$$\int_1^4 \frac{1}{8}\mathrm{x}^2\mathrm{dx} \approx \mathrm{f}\left(\frac{1+7/4}{2}\right)\left(\frac{3}{4}\right) + \mathrm{f}\left(\frac{7/4+10/4}{2}\right)\left(\frac{3}{4}\right) + f\left(\frac{10/4+13/4}{2}\right)\left(\frac{3}{4}\right) + f\left(\frac{13/4+4}{2}\right)\left(\frac{3}{4}\right) =$$

$$= \frac{3}{4}\left[f\left(\frac{11}{8}\right) + f\left(\frac{17}{8}\right) + f\left(\frac{23}{8}\right) + f\left(\frac{29}{8}\right)\right] = \frac{3}{4}[0.2363 + 0.5644 + 1.03 + 1.64] = 2.605$$

$$\int_1^4 \frac{1}{8}\mathrm{x}^2\mathrm{dx} \approx 2.605 \qquad \int_1^4 \frac{1}{8}\mathrm{x}^2\mathrm{dx} = 2.625$$

The midpoint rule



OUTLINE

BASICS

SCATTERING THEORY

- Scattering principles
- Rayleigh and Mie Scattering
- The Phase Function
- The Out-Scattering Equation
- The In-Scattering Equation

CAVEATS OF NAIVE IMPLEMENTATION

ACTUAL IMPLEMENTATION

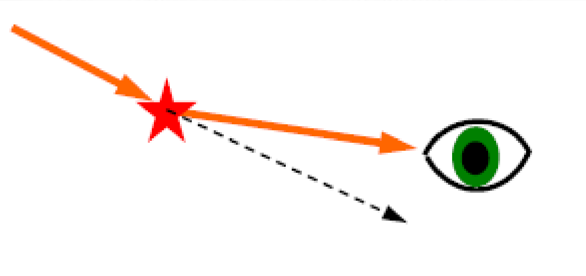
DEMO

CONCLUSIONS AND FUTURE WORK

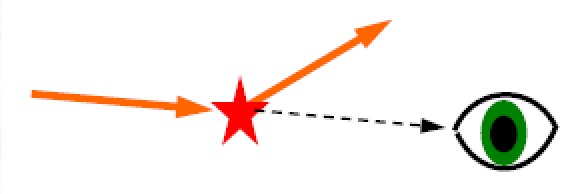
REFERENCES

Scattering events at each sample:

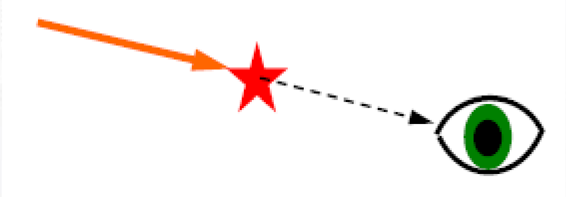
- Light scattered into the line of sight (in-scattering)



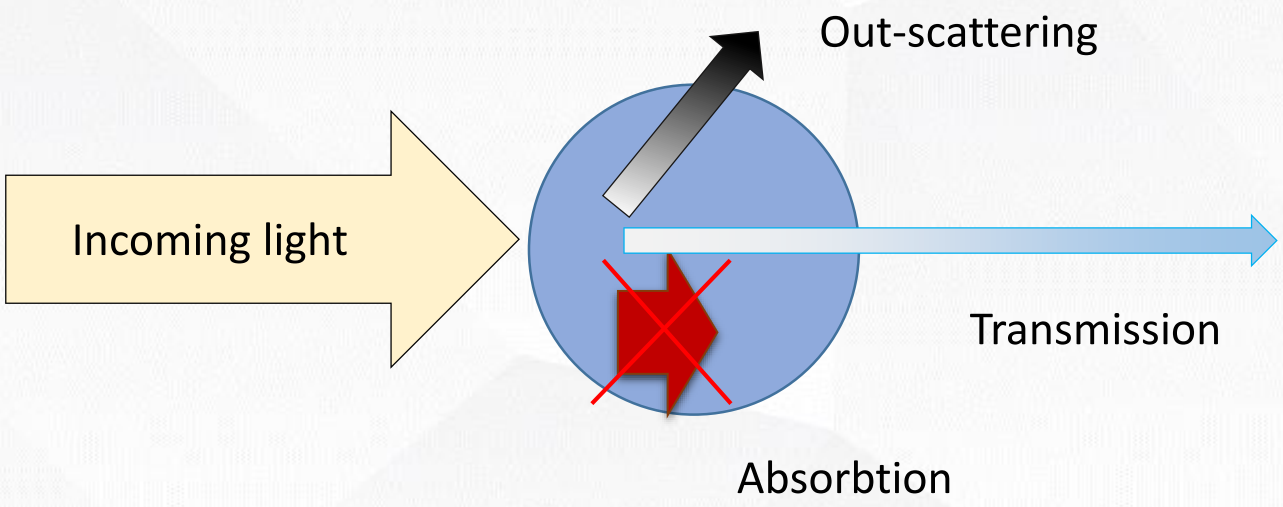
- Light scattered out of the line of sight (out-scattering)



- Light absorbed altogether (absortion)

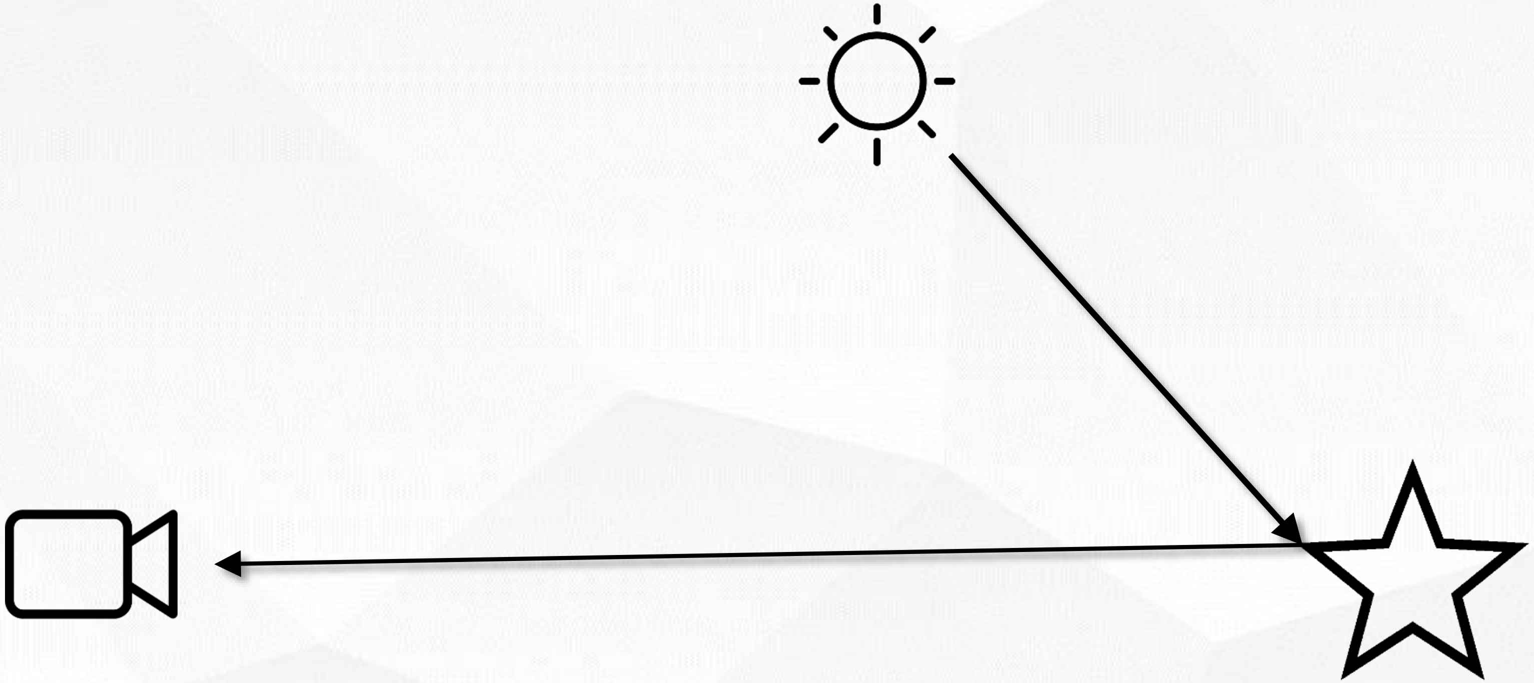


Scattering Principles. Atmospheric scattering

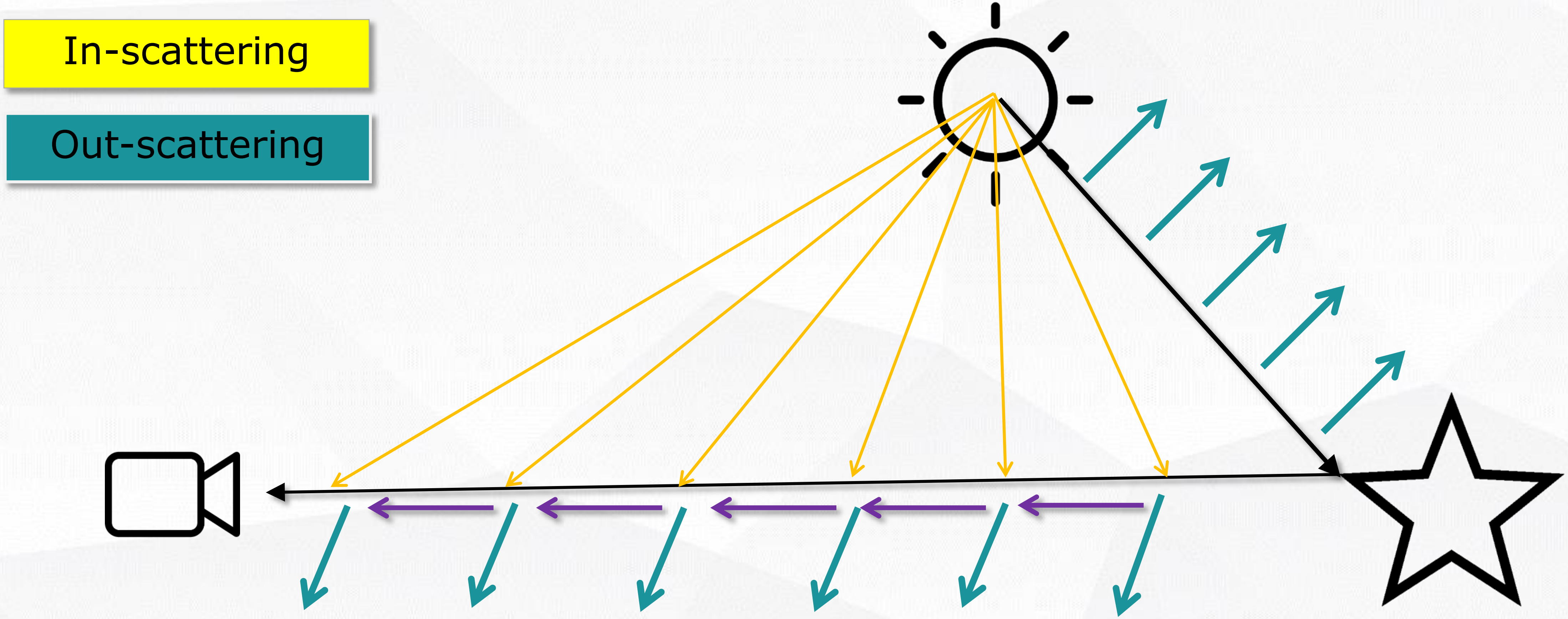


$$L_{incoming} = L_{transmitted} + L_{absorbed} + L_{scattered}$$

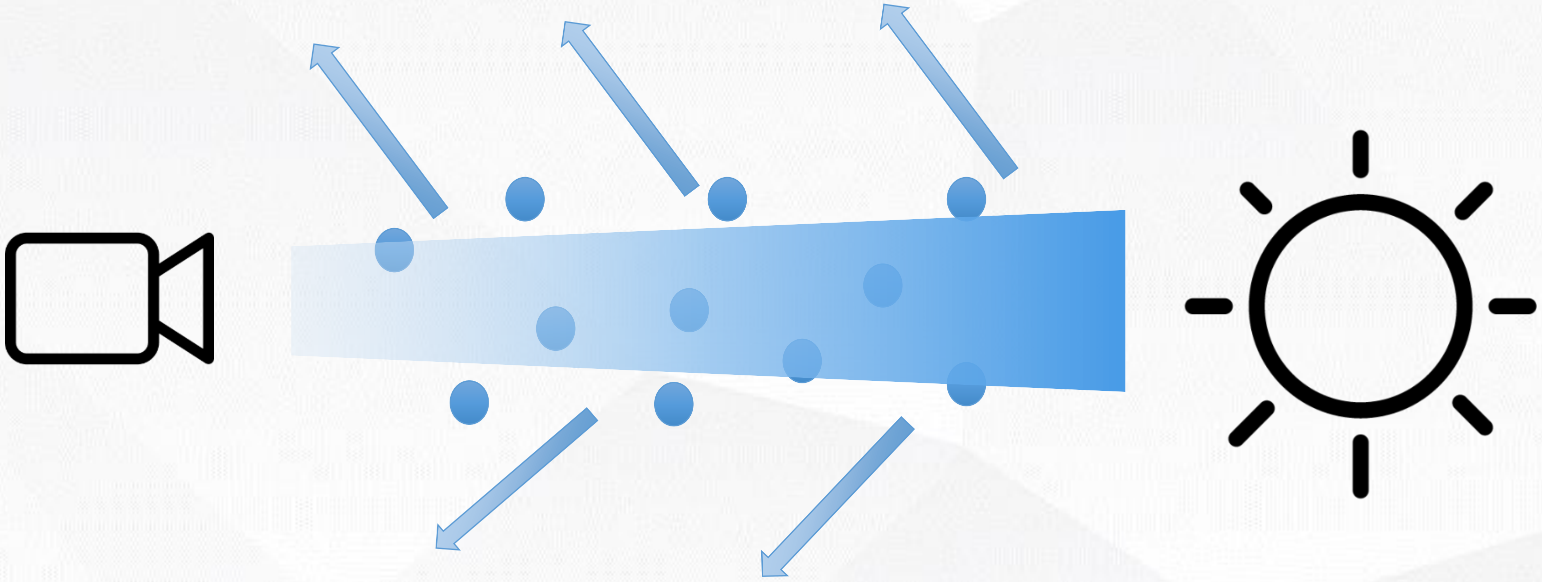
Scattering Principles. No scattering



Scattering Principles. Light Scattering



Scattering Principles. Beer-Lambert Law

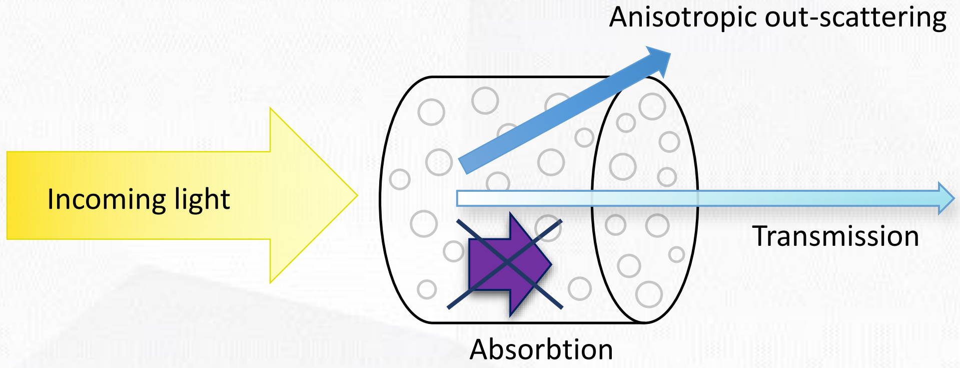


$$T(A \rightarrow B) = e^{-\int_A^B \beta e(x)\,dx}$$

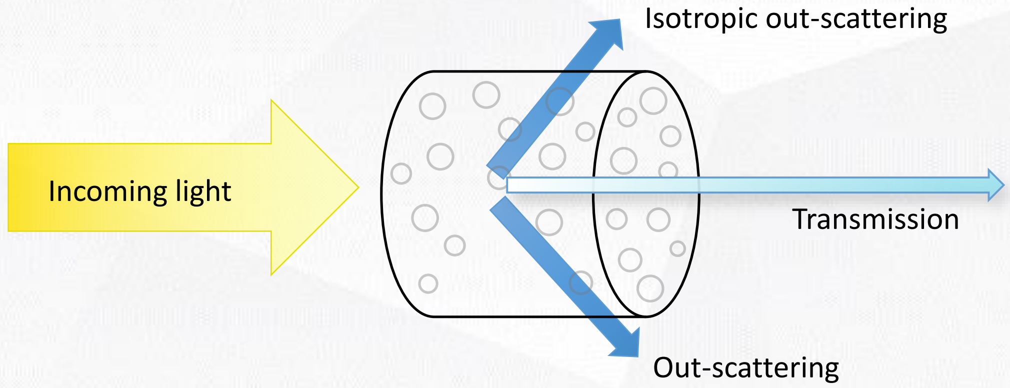
Rayleigh and Mie Scaterring

- Different particles scatter light in diferent ways.
- Most comon types of scatering: **Rayleigh** and **Mie** scattering

Mie scattering

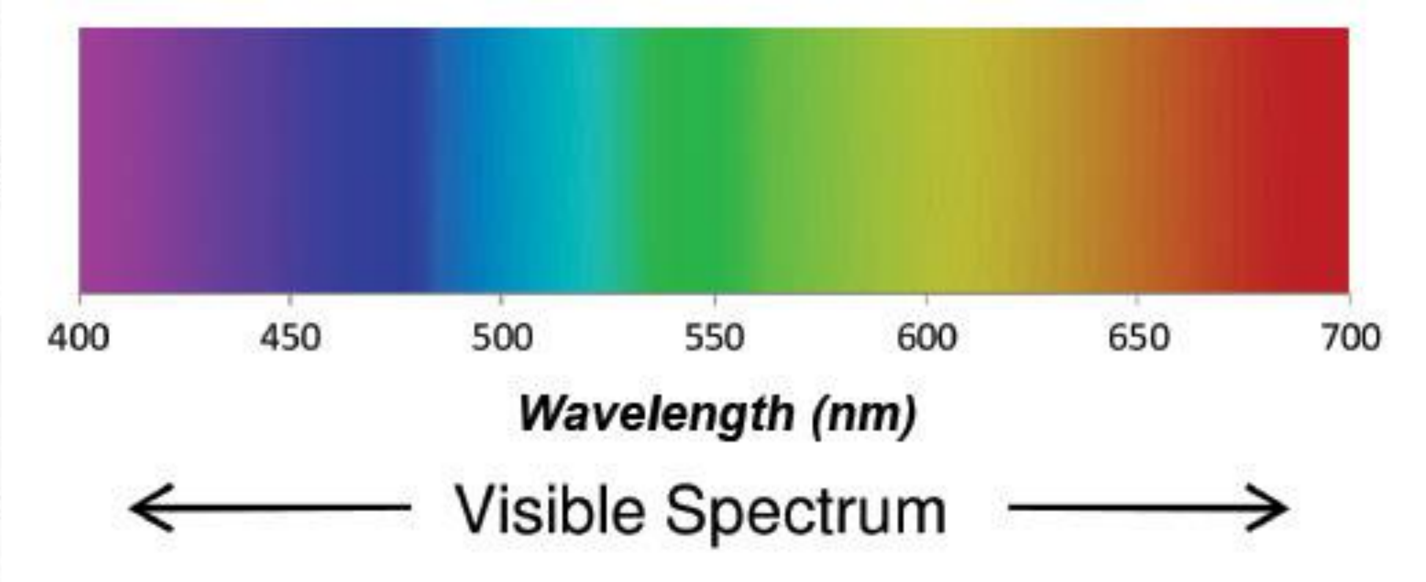


Rayleigh scattering

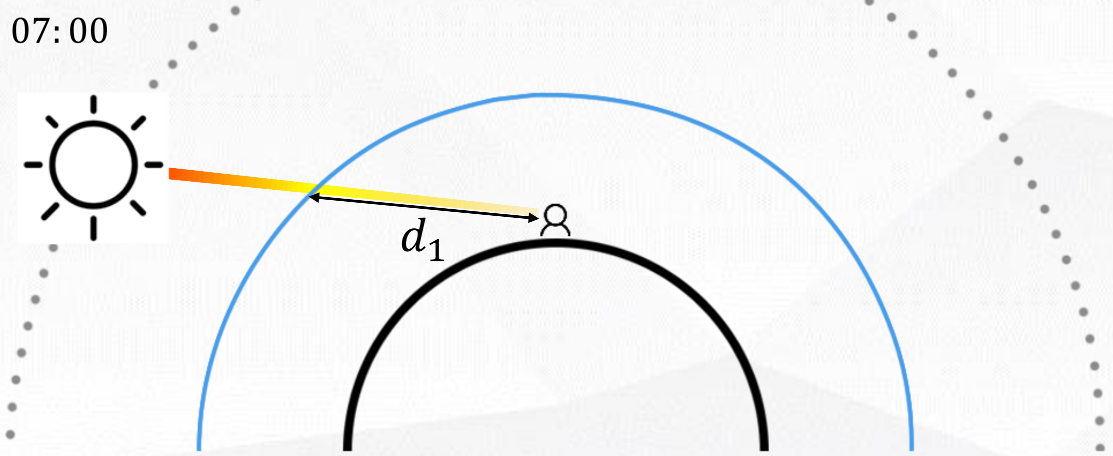


Rayleigh Scaterring

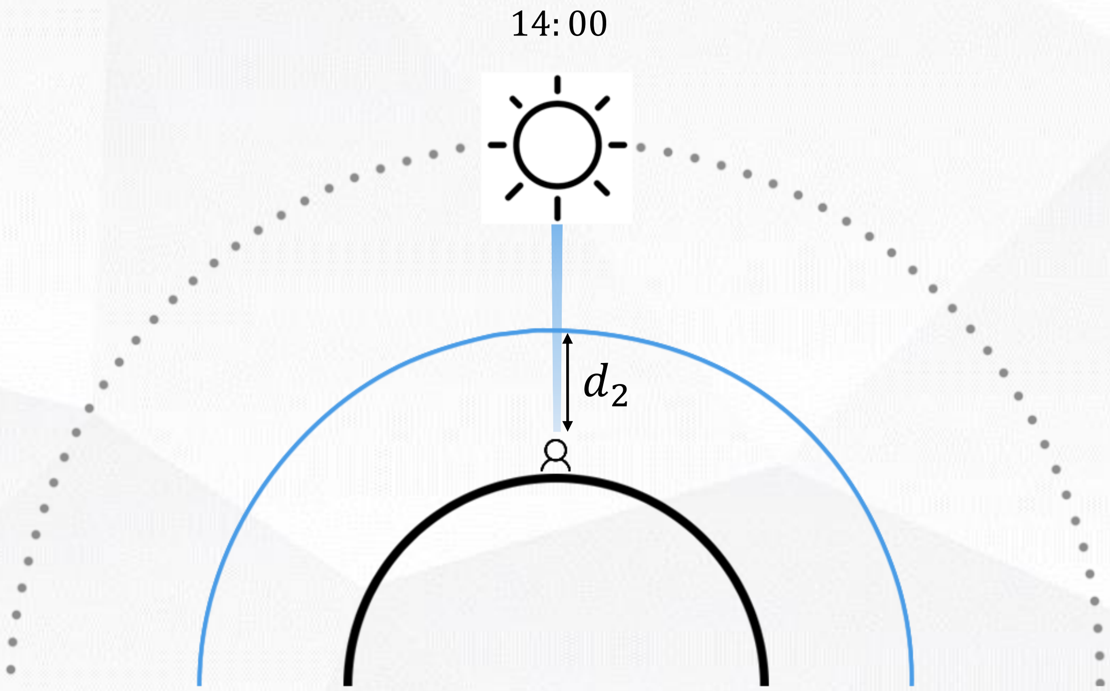
- Caused by small molecules in the air
- Scaters more heavily at shorter wavelengths (blue first, then geren and finally red)
- Why is the sky blue then? Why is it red at sunset?



Dawn & dusk



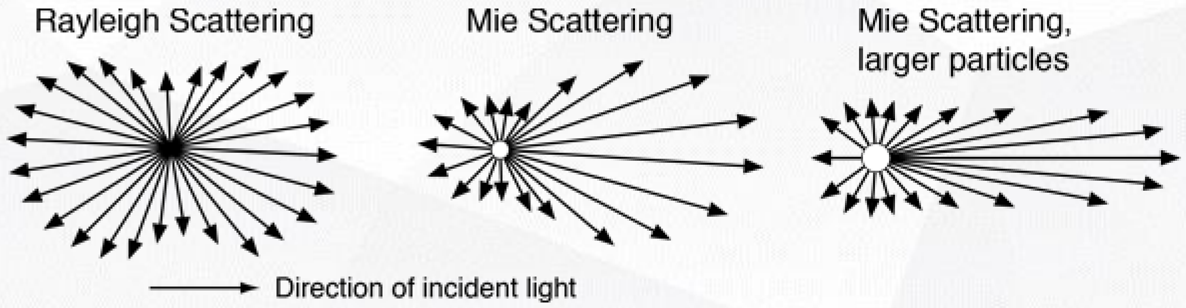
Middle of day



$$d_1 \gg d_2$$

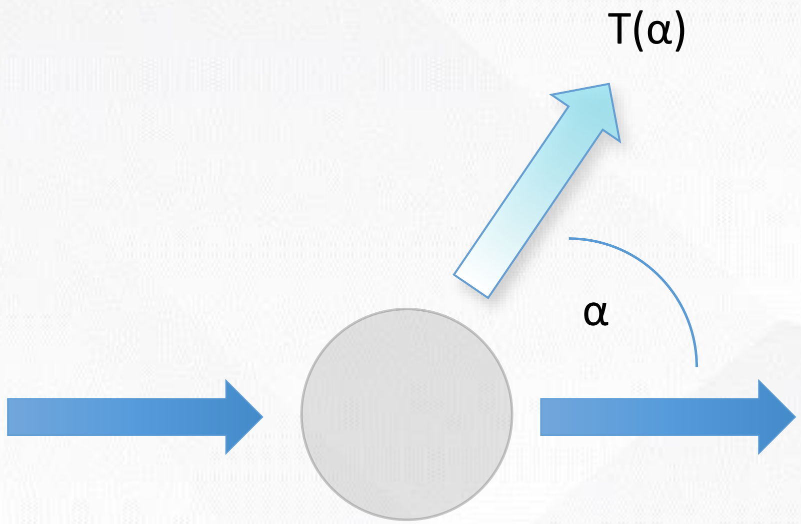
Mie Scaterring

- Caused by larger particles called aerosols (such as dust and pollution)
- Tends to scatter all wavelengths equally
- Highly directional on forward lobe (anisotropic)
- High absorption proportion
- Produces halo on sun on hazy days



The phase function

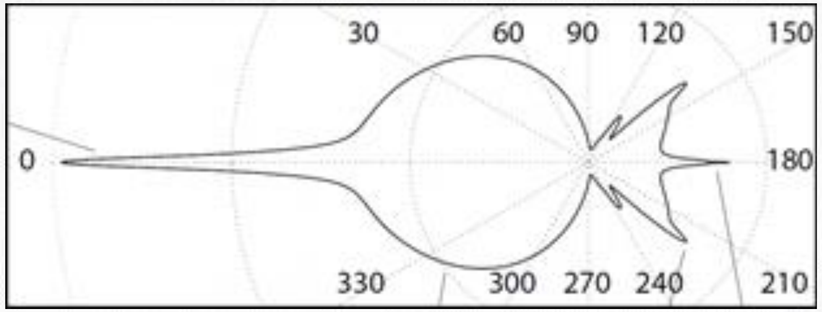
- **Describes how much light is scattered toward all directions**, in the analytical version it depends on angle and a constant g (for directionality)



Light scattered in direction T(α)

Energy conserving

$$\int_0^{2\pi}\int_0^{\pi} P(\theta)\, d\theta\, d\varphi = 1$$



Phase functions can be very complex

Example: clouds phase function

Source: Bouthors et al, “Real-time realistic illumination and shading of stratiform clouds”

Analytical phase function

$$F(\theta, g) = \frac{3 \times (1 - g^2)}{2 \times (2 + g^2)} \times \frac{1 + \cos^2 \theta}{(1 + g^2 - 2 \times g \times \cos\theta)^{\frac{3}{2}}}$$

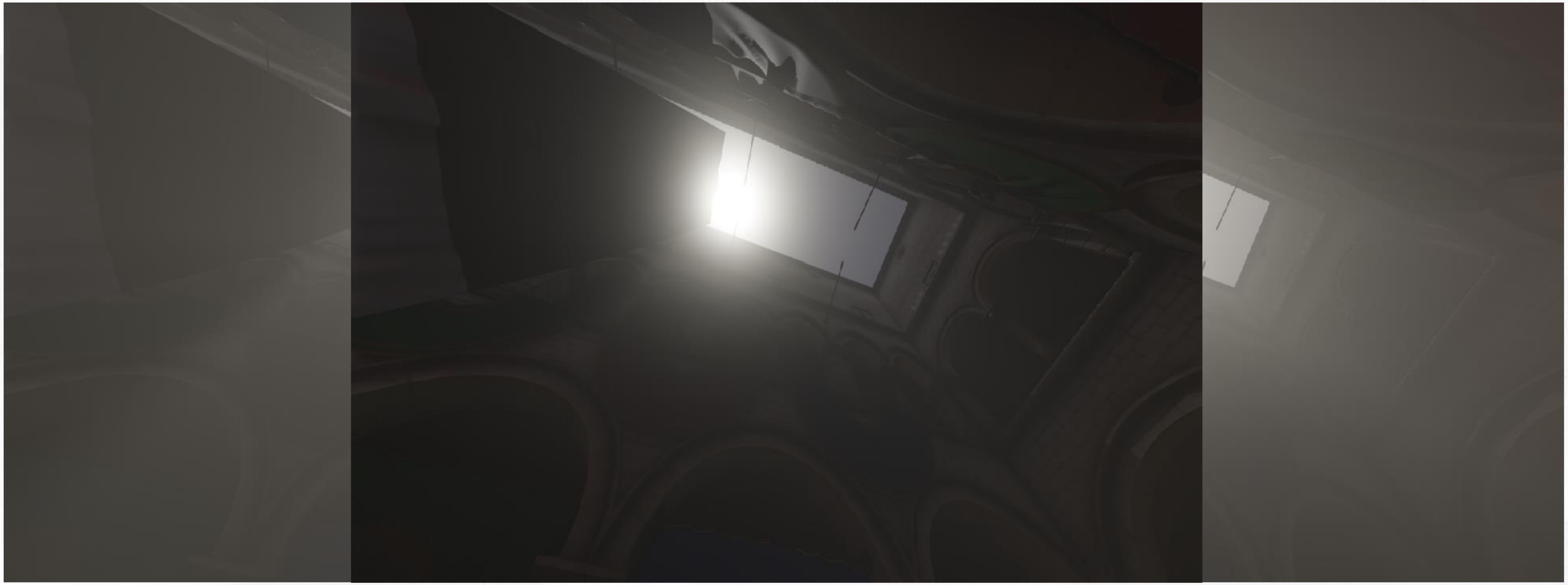
Adaptation of the ***Henyey-Greenstein*** *function used in Nishita et al. 1993.*

g *is the anisotropy factor, for defining directionality of scattering*

$\boldsymbol{Rayleigh\ Scattering}$: $\boldsymbol{g = 0}$ $(Reduces\ complexity, makes\ it\ simetrical)$

$\boldsymbol{Mie\ Scattering}$: $\boldsymbol{g = [-0.75, -0.99]}$ $(Scatters\ more\ light\ in\ forward\ direction)$

Scattering anisotropy



g = 0 g = 0.9 g = 0.3

Negative g *values scatters more light in the* ***forward direction***
Positive g *values scatter more light toward the* ***light source***

The Out-Scattering equation

- Determines **how much light gets scattered away** from the ray **by the particles** it passes

$$t(P_a P_b, \lambda) = 4\pi \times K(\lambda) \times \int_{P_a}^{P_b} \exp\left(\frac{-h}{H_0}\right) ds$$

Wavelength
(or color) of light

Scattering constant
(Rayleigh, Mie scattering)

Optical Depth or ***Optical Thickness***
Average atmospheric density across the ray from P_a to P_b **(quantity of particles)**

Out-Scattering->Optical Depth. In detail

- Think of it as a weighting factor based on how many air particles are in the path of the light along the ray

$$Optical\ Depth = \int_{P_a}^{P_b} exp\left(\frac{-h}{H_o}\right) ds$$

- Will be calculated by broken up the integral into segments
- Exponential evaluated at each sample point $\boldsymbol{h}$ is the height of the sample point (In this implementation $\boldsymbol{h} = \boldsymbol{0}$ is sea level and $\boldsymbol{h} = \boldsymbol{1}$ is top of the atmosphere)
- $\boldsymbol{H_o}$ is the scale height, height at which the atmosphere average density is found. In implementation **= 0.25**, 25% on the way up from ground to top of atmosphere

The In-Scaterring equation

- Determines **how much light is added to a ray** through the atmosphere due to light scattering from the sun
- Main computation for determining sky colour

$$I_v(\lambda) = \underbrace{I_s(\lambda) \times K(\lambda) \times F(\theta, g)}_{\textit{Light scaling functions}} \times \underbrace{\int_{P_a}^{P_b} \left(\exp\left(\frac{-h}{H_0}\right) \times \exp\left(-t(PP_c, \lambda) - t(PP_a, \lambda)\right) \right) ds}_{\textit{Light that travessed the atmosphere}}$$

Light scaling functions

Light that travessed the atmosphere is scaled by this functions

Light that travessed the atmosphere

Final light that gets into the camera after scattering events inside atmosphere

In-Scaterring->Integral. In Detail

$$I_v(\lambda) = I_s(\lambda) \times K(\lambda) \times F(\theta, g) \times \overbrace{\boxed{\int_{P_a}^{P_b} \left(\exp\left(\frac{-h}{H_0}\right) \times \exp\left(-t(PP_c, \lambda) - t(PP_a, \lambda)\right) \right) ds}}^{A}$$

Integral that travesses the ray
P_a starting entry point to atmosphere,
P_b ending entry point from atmosphere

Transmitance (light outscattering)
Proportion of light transported through medium to incoming light from given direction

$$A = \int_{P_a}^{P_b} \left(exp\left(\frac{-h}{H_o}\right) \times exp\left(-t(PP_c, \lambda) - t(PP_a, \lambda)\right) \right) ds$$

Light that travessed the armosphere

Optical Depth
Particle density in ray traversing atmosphere
(NESTED INTEGRAL)

out − scattering point to sun
Light scattered out of ray from point to sun

out − scattering point to camera
Light scattered out of ray from point to camera

In-Scaterring->Scaling light. In Detail

$$I_v(\lambda) = \overset{B}{\boxed{I_s(\lambda) \times K(\lambda) \times F(\theta, g)}} \times \int_{P_a}^{P_b} \left(\exp\left(\frac{-h}{H_0}\right) \times \exp\left(-t(PP_c, \lambda) - t(PP_a, \lambda)\right) \right) ds$$

$$\boldsymbol{B} = I_s(\lambda) \times K(\lambda) \times F(\theta, g)$$

Sunlight intensity
scaling the incoming light by the sun intensity *(depending of wavelength for snazzy alien effects)*

Scattering constant
scaling the incoming light by the scattering constants (Rayleigh & Mie)

Phase Function
Directionality of the final light going to camera

The In-Scaterring equation. Complete

Integral that travesses the ray
P_a starting entry point to atmosphere,
P_b ending entry point from atmosphere

Transmitance (light outscattering)
Proportion of light transported through medium to incoming light from given direction

$$I_v(\lambda) = I_s(\lambda) \times K(\lambda) \times F(\theta, g) \times \int_{P_a}^{P_b} \left(\exp\left(\frac{-h}{H_0}\right) \times \exp\left(-t(PP_c, \lambda) - t(PP_a, \lambda)\right)\right) ds$$

Sunlight intensity
scaling the incoming light by the sun intensity *(depending of wavelength for snazzy alien effects)*

Scattering constant
scaling the incoming light by the scattering constants (Rayleigh & Mie)

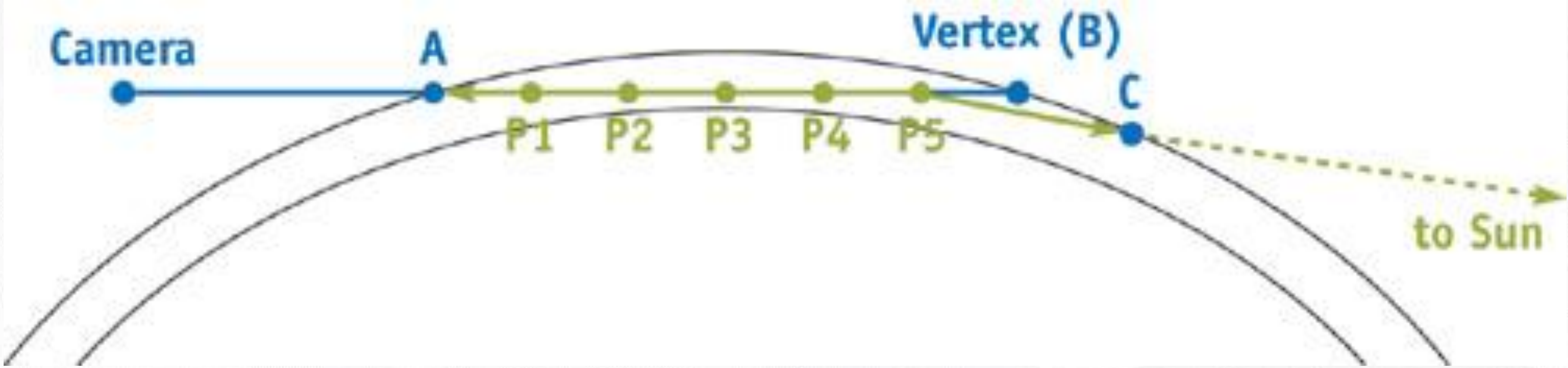
Phase Function
Directionality of the final light going to camera

optical depth
Particle density in ray traversing atmosphere
(NESTED INTEGRAL)

out − scattering point to sun
Light scattered out of ray from point to sun

out − scattering point to camera
Light scattered out of ray from point to camera

Visual Representation



OUTLINE

Naive Implementation (1)

For example:

- 5 sample points in the in-scattering equation
- 5 sample points for each of the integrals to compute out-scattering equations

$$I_v(\lambda) = I_s(\lambda) \times k(\lambda) \times F(\theta,\lambda) \times \sum_{i=0}^{N=5} \left(exp\left(\frac{-h}{H_o}\right) \times exp\big(-t(PP_c,\lambda) - t(PP_a,\lambda)\big) \right) ds$$

$$t(P_aP_b,\lambda) = 4\pi \times k(\lambda) \times \sum_{i=0}^{N=5} exp\left(\frac{-h}{H_o}\right) ds$$

$N_{operations} \approx 5 \times (5+5)$ samples => To evaluate at every vertex (doing it in vertex shader pass)

→ Two types of scattering, Rayleigh and Mie

→ Each one of the different wavelengths of each color channel (RGB)

SO…

Naive Implementation (2)

$$N_{operations} \approx 2 \times 3 \times 5 \times (5 + 5) = \text{300 computations per vertex}$$

Five samples is not enough. Low quality. O’Neil uses **50 samples for inner integral** and **5 samples for outer integral**

$$N_{operations} \approx 2 \times 3 \times 5 \times (50 + 50) = \text{3000 computations per vertex!!!}$$

Making it Real-Time

- To cut number of calculations O'Neill proposed **a 2D lookup table** where the **results of both integrals are stored** using 50 samples for each. Calculated beforehand on CPU

- A lookup table is a texture used to save data that can be fetched in a shader instead of doing the same calculation in the shader pass

- The lookup table permits to cut number of calculations to: $2 \times 3 \times 5 \times (1 + 1) = 60$

- Enough for 60 to 100 frames per second by 2007 standards

Making it Real-Time

- O'Neil wants to eliminate the sampling of textures in vertex shader
- Eliminate the 2D lookup table by finding heuristic equations that approximate the results of the integral
- Finds the heuristic by analyzing the results of the lookup table integrals.

- *First approximation* :

  Optical Depth at point approximated by using $\boldsymbol{exp(-4h)}$ instead of $\boldsymbol{exp\left(\frac{-h}{H_o}\right)}$

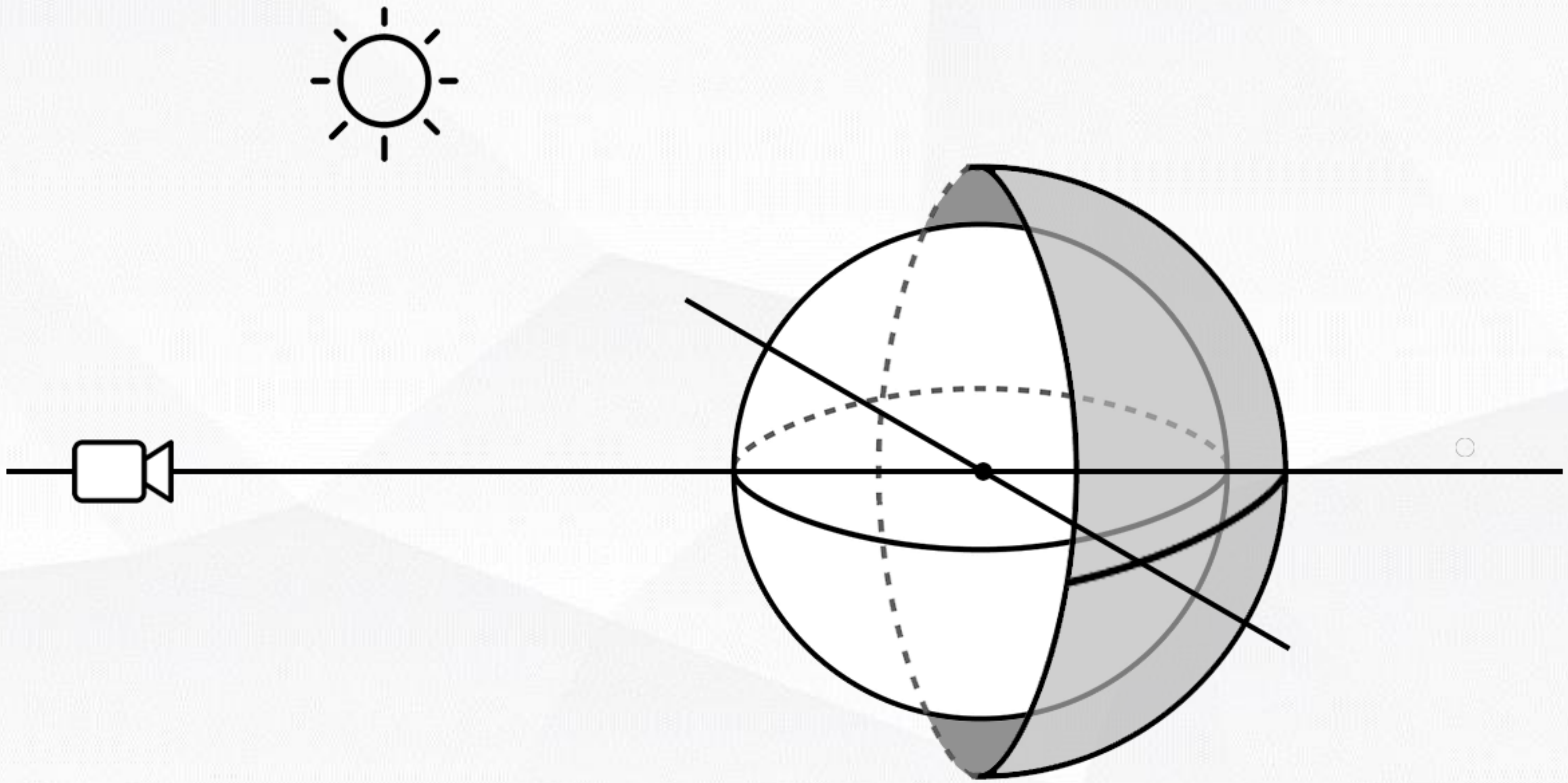
- *Second approximation* :

  Optical Depth integral approximated by using a **special scale function** that uses a polynomial. This function obligues use to have the atmosphere be 2.5% of the planet's radius, and the scale height (height where the atmosphere's average density is found) = 0.25

OUTLINE

Geometry

- Two spheres, atmosphere and planet
- Atmosphere is sphere only showing the back faces. (Culling to remove front faces)
- Free floating camera. Vector defining position of sun in world space



Vertex Shader

- Majority of calculations done in vertex shader pass
- Some calculations done at fragment shader pass to avoid artifacts. (Phase function)

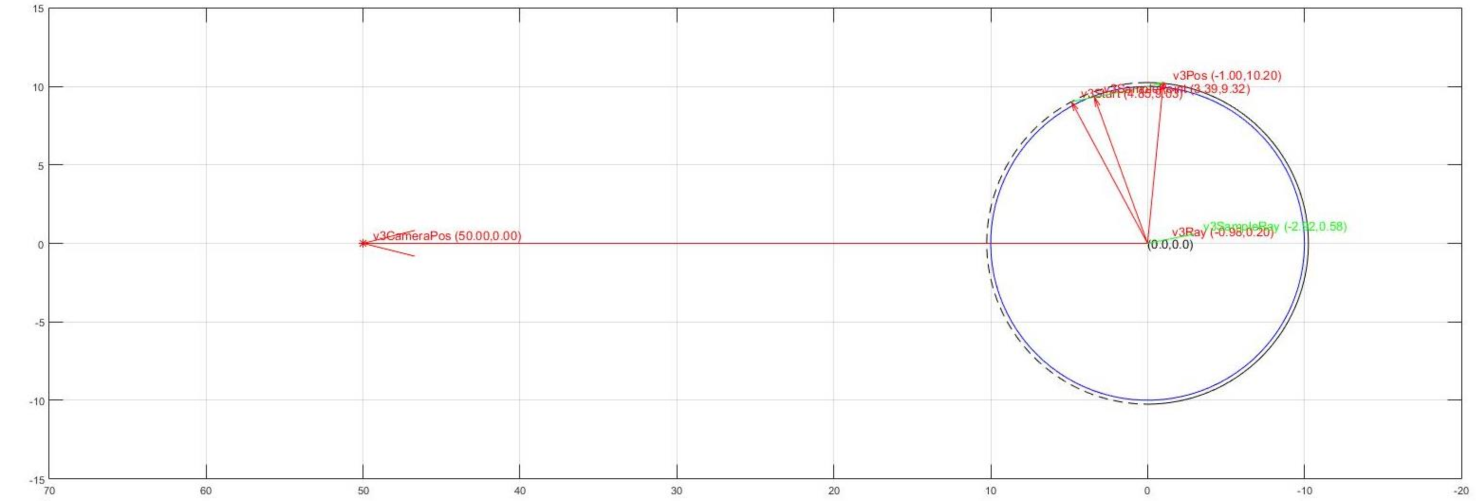
UNIFORMS:

- Kr rayleigh scattering constant
- Km mie scattering constant
- Esun is the brightness of the sun
- Rayleigh scatters different wavelengths at different ratios, ratio = $^{1}/_{wavelength^4}$

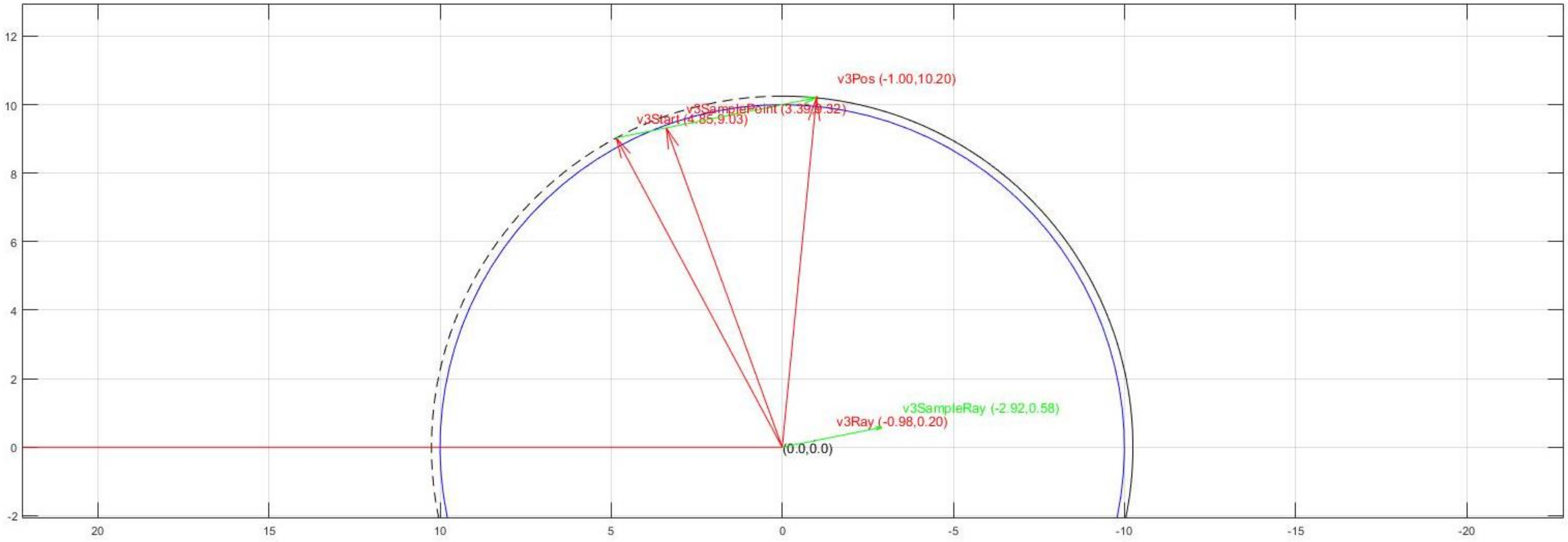
Vertex Shader. SkyFromAtmosphere-vs

```
attribute vec3 aVertexPosition;

varying vec3 v3Direction;
varying vec3 frontColor;
varying vec3 frontSecondaryColor;

varying vec3 debugColor;

uniform mat4 uModelMatrix;
uniform mat4 uViewMatrix;
uniform mat4 uProjectionMatrix;

uniform vec3 v3CameraPos;              // The camera's current position

uniform vec3 v3LightPos;               // The direction vector to the light source
uniform vec3 v3InvWavelength;          // 1 / pow(wavelength, 4) for the red, green, and blue channels
uniform float fCameraHeight;           // The camera's current height

uniform float fCameraHeight2;          // fCameraHeight^2
uniform float fOuterRadius;            // The outer (atmosphere) radius
uniform float fOuterRadius2;           // fOuterRadius^2

uniform float fInnerRadius;            // The inner (planetary) radius
uniform float fInnerRadius2;           // fInnerRadius^2
uniform float fKrESun;                 // Kr * ESun

uniform float fKmESun;                 // Km * Esun
uniform float fKr4PI;                  // Kr * 4 * PI
uniform float fKm4PI;                  // Km * 4 * PI

uniform float fScale;                  // 1 / (fOuterRadius - fInnerRadius)
uniform float fScaleDepth;             // The scale depth (i.e. the altitude at which the atmosphere's average density is found)
uniform float fScaleOverScaleDepth;    // fScale / fScaleDepth

uniform sampler2D uTextureDebug;
uniform sampler2D uOpticalDepthLUT;

#define nSamples 2

//const int nSamples = 2;
```

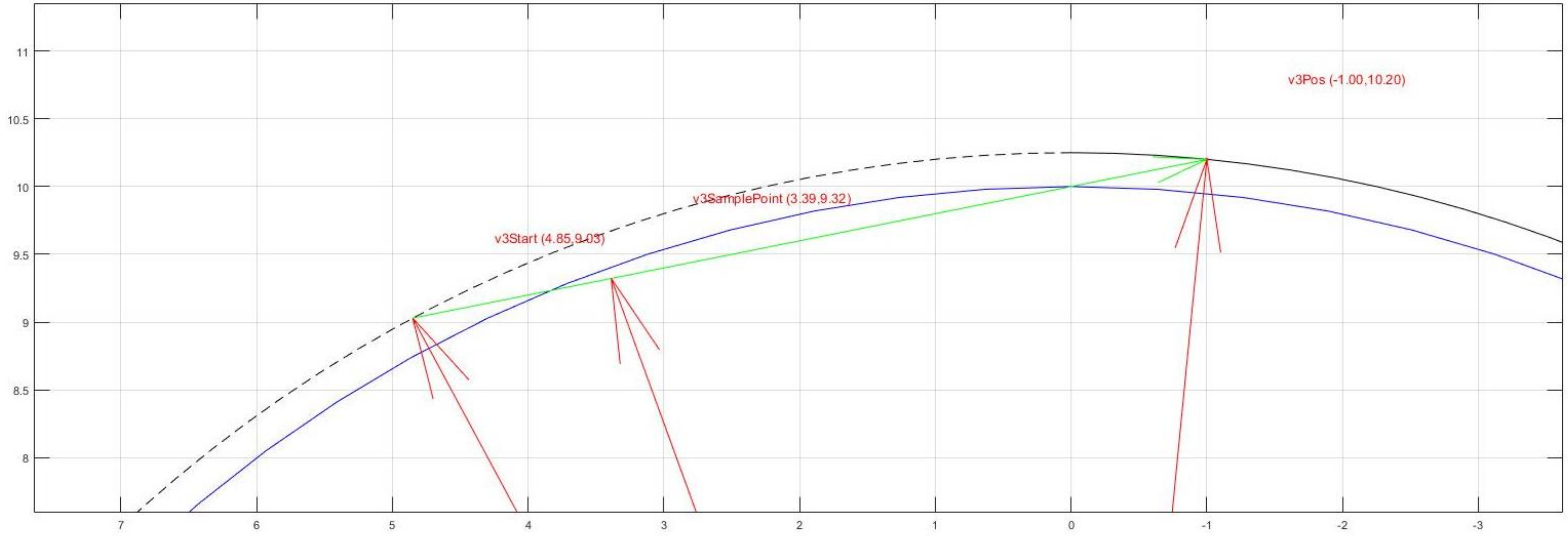
```
#define nSamples 2

//const int nSamples = 2;
const float fSamples = 2.0;

float scale(float fCos)
{
    float x = 1.0 - fCos;
    return fScaleDepth * exp(-0.00287 + x*(0.459 + x*(3.83 + x*(-6.80 + x*5.25))));
}

void main(void)
{
    // Get the ray from the camera to the vertex and its length
    vec4 aux = uModelMatrix * vec4(aVertexPosition, 1.0);
    vec3 v3Pos = aux.xyz;
    vec3 v3Ray = v3Pos - v3CameraPos;
    float fFar = length(v3Ray);
    v3Ray /= fFar;

    // Calculate the ray's starting position, then calculate its scattering offset
    vec3 v3Start = v3CameraPos;
    float fHeight = length(v3Start);
    float fDepth = exp(fScaleOverScaleDepth * (fInnerRadius - fCameraHeight));

    float fStartAngle = dot(v3Ray, v3Start) / fHeight;
    float fStartOffset = fDepth * scale(fStartAngle);

    vec4 v4LightDepth;
    vec4 v4SampleDepth;

    // Initialize the scattering loop variables
    vec3 v3RayleighSum = vec3(0.0,0.0,0.0);
    vec3 v3MieSum = vec3(0.0,0.0,0.0);
    vec3 v3Attenuation;
    float fSampleLength = fFar / fSamples;
    float fScaledLength = fSampleLength * fScale;
    vec3 v3SampleRay = v3Ray * fSampleLength;
    vec3 v3SamplePoint = v3Start + v3SampleRay * 0.5;
```



v3Pos (-1.00,10.20)
v3Start (4.85,9.03)
v3SamplePoint (3.39,9.32)
v3CameraPos (50.00,0.00)
v3SampleRay (-2.92,0.58)
v3Ray (-0.98,0.20)
(0.0,0.0)

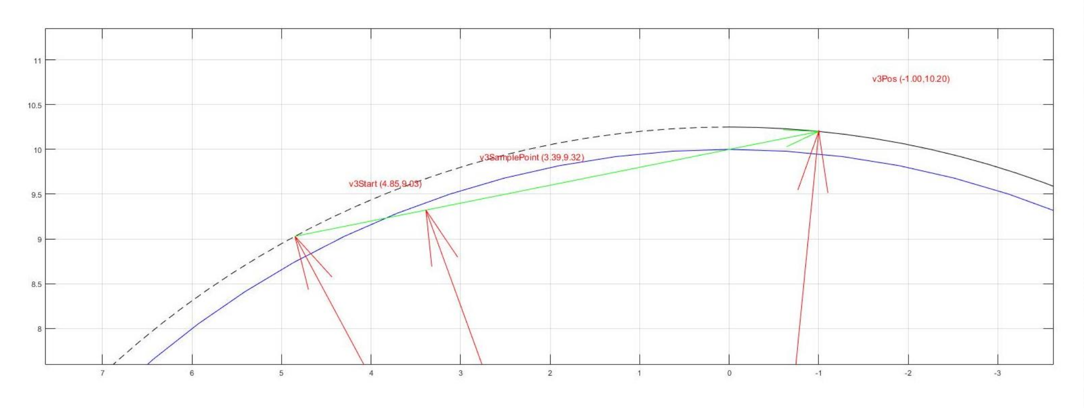


v3Pos (-1.00,10.20)
v3SamplePoint (3.39,9.32)
v3Start (4.85,9.03)
v3SampleRay (-2.92,0.58)
v3Ray (-0.98,0.20)
(0.0,0.0)



v3Pos (-1.00,10.20)
v3SamplePoint (3.39,9.32)
v3Start (4.85,9.03)
11
10.5
10
9.5
9
8.5
8
7
6
5
4
3
2
1
0
-1
-2
-3

```
    vec3 v3SamplePoint = v3Start + v3SampleRay * 0.5;

    // Now loop through the sample rays
    vec3 v3FrontColor = vec3(0.0, 0.0, 0.0);

    for (int i=0; i<nSamples; i++)
    {
        float fHeight = length(v3SamplePoint);
        float fDepth = exp(fScaleOverScaleDepth * (fInnerRadius - fHeight));

        float fLightAngle = dot(v3LightPos, v3SamplePoint) / fHeight;
        float fCameraAngle = dot(v3Ray, v3SamplePoint) / fHeight;

        float fScatter = (fStartOffset + fDepth*(scale(fLightAngle) - scale(fCameraAngle)));

        vec3 v3Attenuate = exp(-fScatter * (v3InvWavelength * fKr4PI + fKm4PI));

        v3FrontColor += v3Attenuate * (fDepth * fScaledLength);

        v3SamplePoint += v3SampleRay;
    }

    debugColor = v3FrontColor * (v3InvWavelength * fKrESun);

    // Finally, scale the Mie and Rayleigh colors and set up the varying variables for the pixel shader
    frontSecondaryColor.rgb = v3FrontColor * fKmESun; // MieColour
    frontColor.rgb = v3FrontColor * (v3InvWavelength * fKrESun); // RayleighColour

    gl_Position = uProjectionMatrix * uViewMatrix * uModelMatrix * vec4(aVertexPosition, 1.0);
    v3Direction = v3CameraPos - v3Pos;
}
```



Midpoint rule integral approximation

$\left|\overrightarrow{v3SamplePoint}\right|$

```
for (int i=0; i<nSamples; i++)
{
    float fHeight = length(v3SamplePoint);
    float fDepth = exp(fScaleOverScaleDepth * (fInnerRadius - fHeight));
```

$$fScaleOverScaleDepth = \frac{fScale}{fScaleDepth}$$

$$fDepth = \frac{\frac{1}{(fOuterRadius\ - fInnerRadius)}}{\frac{0.25}{1}} \times (fInnerRadius\ - fHeight) = \underbrace{\boldsymbol{exp(-4h)}}_{\text{From first approximation}}$$

```
float fLightAngle = dot(v3LightPos, v3SamplePoint) / fHeight;
float fCameraAngle = dot(v3Ray, v3SamplePoint) / fHeight;
```

$$u \cdot v = |\vec{u}| \times |\vec{v}| \times \cos\alpha \Rightarrow \cos\alpha = \frac{u \cdot v}{|\vec{u}| \times |\vec{v}|}$$

$v3LightPos$ is actually the direction of light (normalized),
$v3Ray$ is also normalized

Out-scattering

$$t(P_a P_b, \lambda) = 4\pi \times k(\lambda) \times \sum_{i=0}^{N} exp\left(\frac{-h}{H_o}\right) ds$$

```
float fScatter = (fStartOffset + fDepth*(scale(fLightAngle) - scale(fCameraAngle)));
```

$$fScatter = \sum_P^{P_c} exp\left(\frac{-h}{H_o}\right) ds + \sum_P^{P_a} exp\left(\frac{-h}{H_o}\right) ds$$

Out-scattering

$$t(P_aP_b,\lambda) = 4\pi \times k(\lambda) \times \sum_{i=0}^{N} exp\left(\frac{-h}{H_o}\right) ds$$

In-scattering

$$I_v(\lambda) = I_s(\lambda) \times k(\lambda) \times F(\theta,\lambda) \times \sum_{i=0}^{N}\left(exp\left(\frac{-h}{H_o}\right) \times exp\left(-t(PP_c,\lambda) - t(PP_a,\lambda)\right)\right) ds$$

```
vec3 v3Attenuate = exp(-fScatter * (v3InvWavelength * fKr4PI + fKm4PI));
```

$$\sum_{P}^{P_c} exp\left(\frac{-h}{H_o}\right) ds + \sum_{P}^{P_a} exp\left(\frac{-h}{H_o}\right) ds$$

Rayleigh scattering coefficient, dependent on wavelength

Mie scattering coeffient

$$v3FrontColor = \sum_{i=0}^{N} \left(exp\left(\frac{-h}{H_o}\right) \times exp\left(-t(PP_c, \lambda) - t(PP_a, \lambda)\right) \right) ds$$

Length of the sampled segment, *midpoint rule*

```
    v3FrontColor += v3Attenuate * (fDepth * fScaledLength);

    v3SamplePoint += v3SampleRay;
}
```

Move towards the next sample point

Finally...

```
    // Finally, scale the Mie and Rayleigh colors and set up the varying variables for the pixel shader
    frontSecondaryColor.rgb = v3FrontColor * fKmESun; // MieColour
    frontColor.rgb = v3FrontColor * (v3InvWavelength * fKrESun); // RayleighColour

    gl_Position = uProjectionMatrix * uViewMatrix * uModelMatrix * vec4(aVertexPosition, 1.0);
    v3Direction = v3CameraPos - v3Pos;
}
```

Fragment Shader

- Phase function multiplication by varying Rayleigh and Mie colors outputted from vs

Fragment Shader. SkyFromAtmosphere-fs

```
precision mediump float;

uniform vec3 v3LightPos;
uniform float g;
uniform float g2;

uniform sampler2D uTextureDebug;

varying vec3 v3Direction;
varying vec3 frontColor;
varying vec3 frontSecondaryColor;

varying vec3 debugColor;

float GetMiePhase(float fCos, float fCos2, float g, float g2);
float GetRayleighPhase(float fCos2);

void main(void)
{

    //gl_FragColor = vec4(0.98, 0.54, 0.035, 1.0); return;
    float fCos = dot(v3LightPos, v3Direction) / length(v3Direction);
    float fCos2 = fCos*fCos;

    vec3 col = GetRayleighPhase(fCos2) *  frontColor + GetMiePhase(fCos, fCos2, g, g2) * frontSecondaryColor;

    gl_FragColor = vec4(col, 1.0); return;
}

float GetMiePhase(float fCos, float fCos2, float g, float g2)
{
    return 1.5 * ((1.0 - g2) / (2.0 + g2)) * (1.0 + fCos2) / pow(1.0 + g2 - 2.0*g*fCos, 1.5);
}

float GetRayleighPhase(float fCos2)
{
    return 0.75 + 0.75 * fCos2;
}
```

OUTLINE

BASICS

SCATERRING THEORY

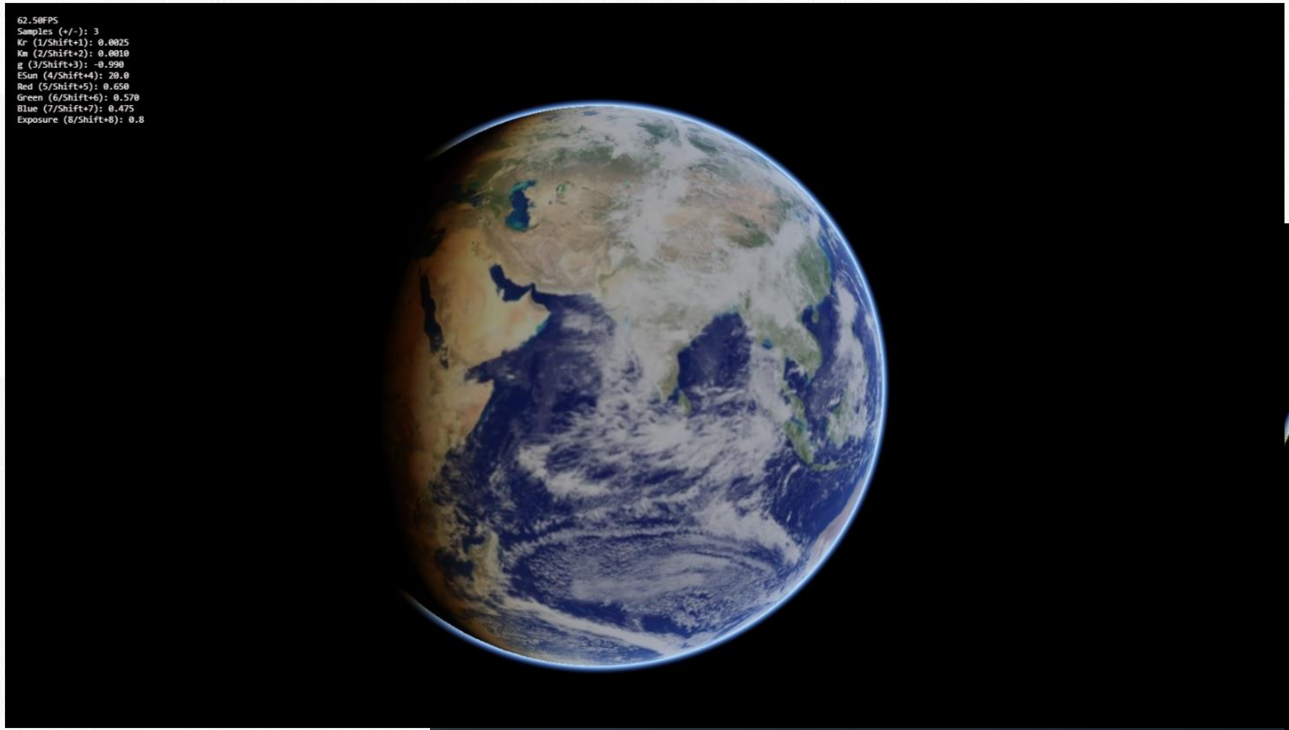
CAVEATS OF NAIVE IMPLEMENTATION

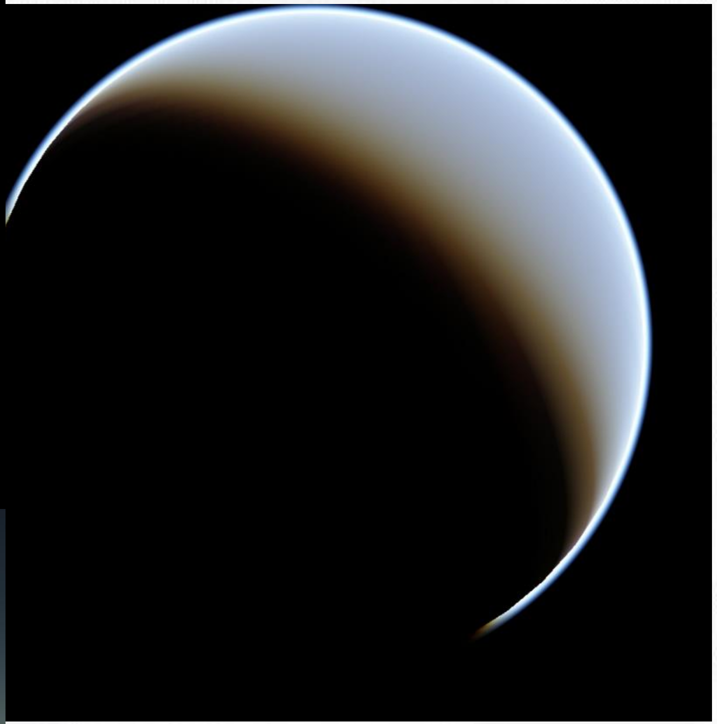
ACTUAL IMPLEMENTATION

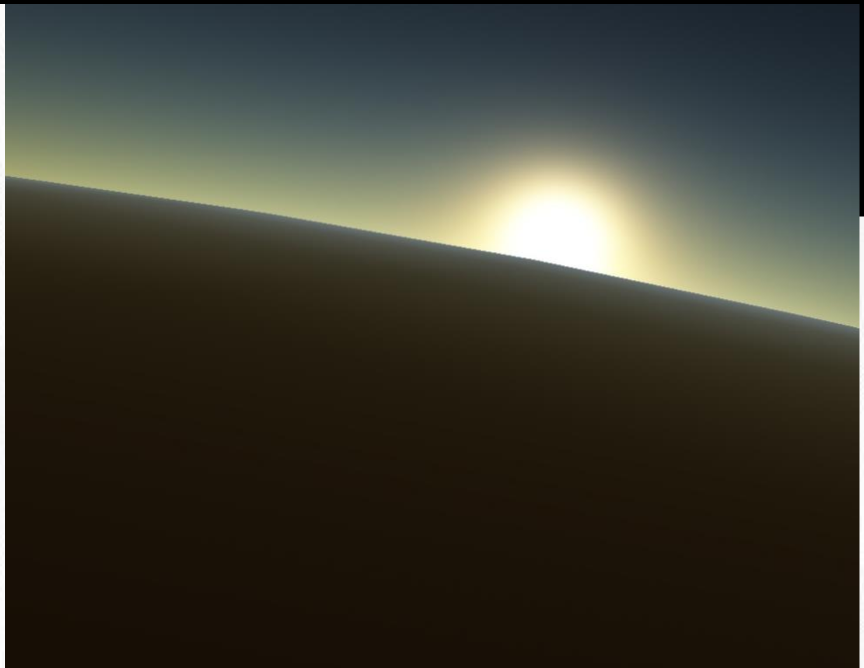
DEMO

CONCLUSIONS AND FUTURE WORK

REFERENCES


62.50FPS
Samples (+/-): 3
Kr (1/Shift+1): 0.0025
Km (2/Shift+2): 0.0010
g (3/Shift+3): -0.990
ESun (4/Shift+4): 20.0
Red (5/Shift+5): 0.650
Green (6/Shift+6): 0.570
Blue (7/Shift+7): 0.475
Exposure (8/Shift+8): 0.8





OUTLINE

Conclusions and Future Work

- Gets results and its fast, could be rendered at a full screen quad instead of sphere, super optimal.
- Caveat of the approximations used, cannot change the atmosphere or density at will

- Add multiple scattering, multiple scattering makes the sky more real
- Remove exponential decrease of atmospheric density approximation
- Scattering from the moon
- Add Atmospheric Perspective
- etc…

OUTLINE

References

- [REF1]

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.5595&rep=rep1&type=pdf

- [REF2]

https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter16.html

- https://github.com/Scrawk/Brunetons-Improved-Atmospheric-Scattering
- https://www.gamedev.net/articles/programming/graphics/real-time-atmospheric-scattering-r2093/

Thank you!

https://github.com/graushf/AtmosphericScatteringWebGL.git

graushf@gmail.com
www.linkedin.com/in/gustavo-raush-faggembauu