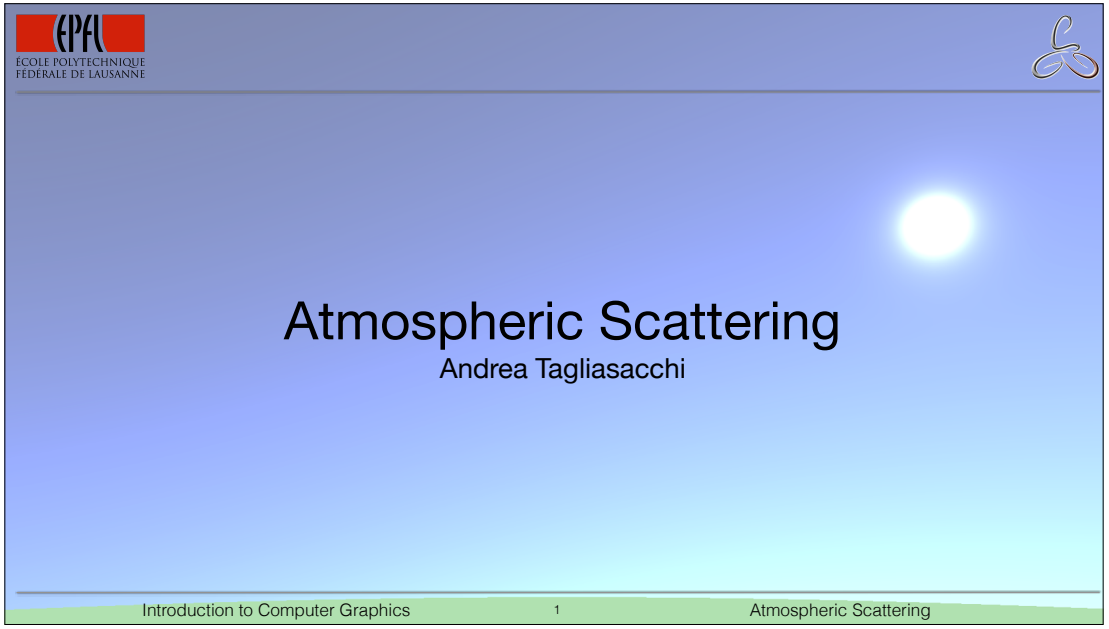
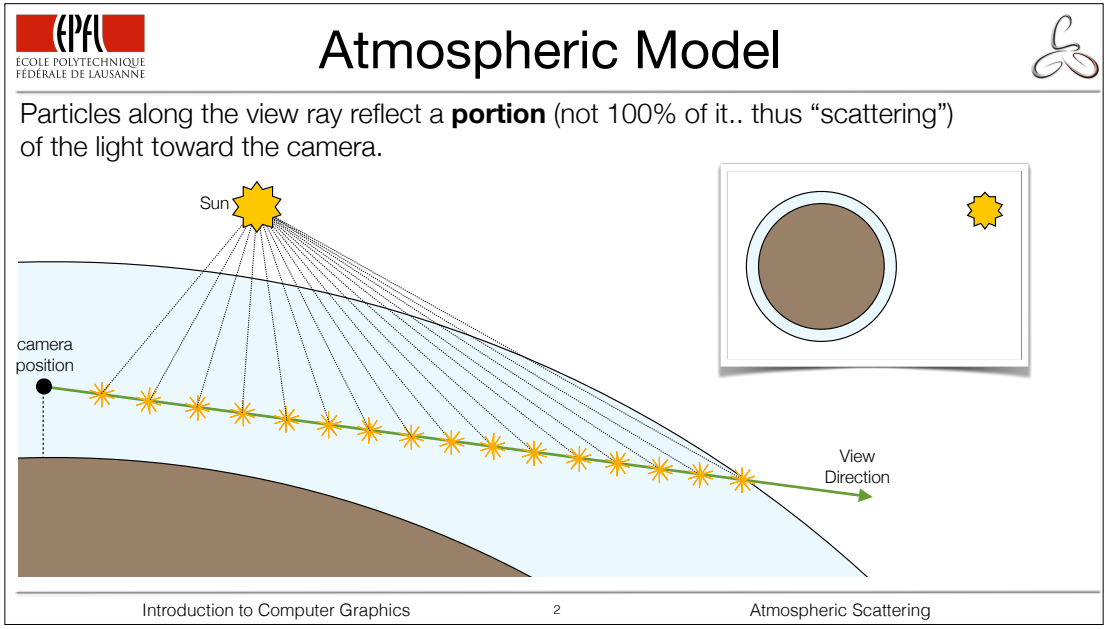


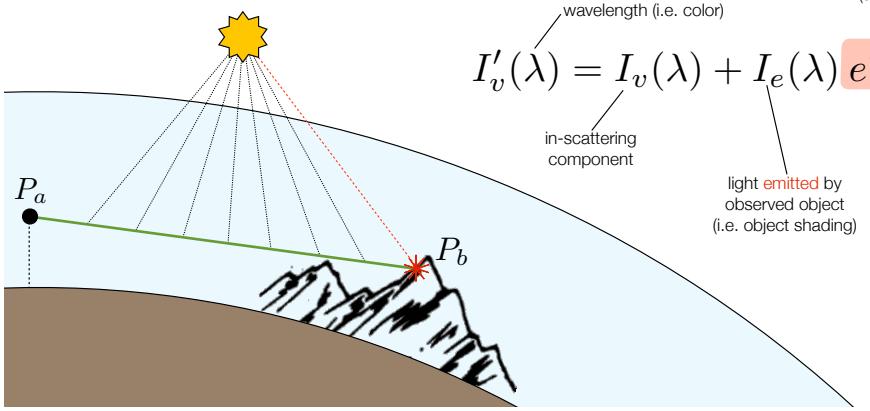
Material prepared from this chapter of GPU Gems 2:
http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter16.html



planet: earth radius + atmosphere radius
observer: height + view direction (angle)
sun: position (or direction assuming far-field approximation)

Are you looking at an object?

we must also take into account that some of the light reflected by the object will be scattered on the way to the camera



wavelength (i.e. color)

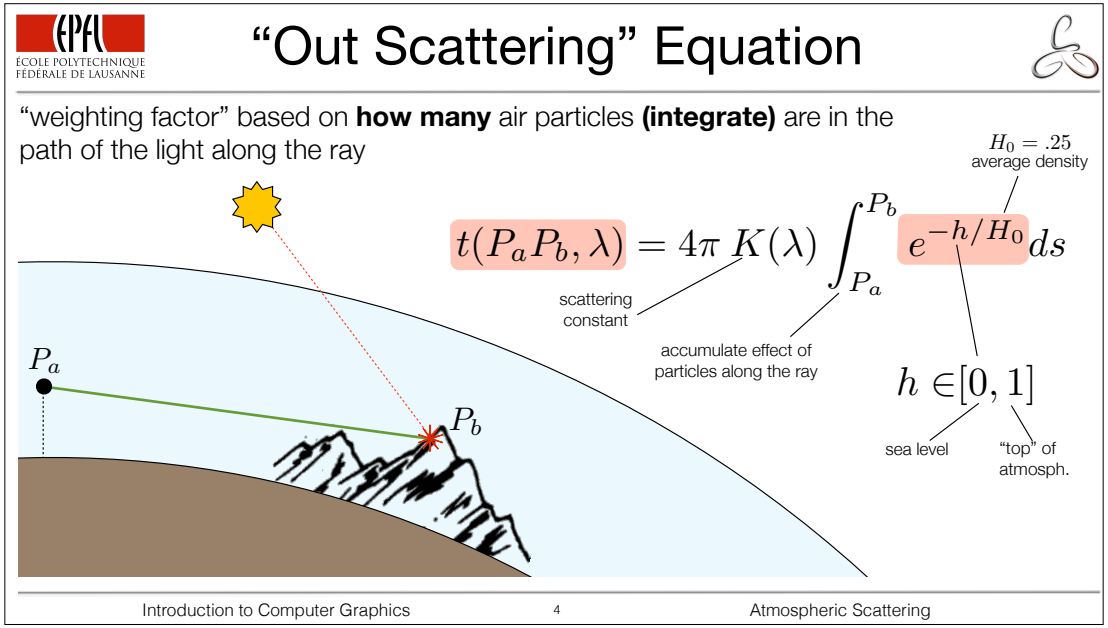
t(a,b): "out scattering" integral

$$I'_v(\lambda) = I_v(\lambda) + I_e(\lambda)\, e^{-t(P_a P_b, \lambda)}$$

in-scattering component

light emitted by observed object (i.e. object shading)

how much light we will be scattered along the view ray

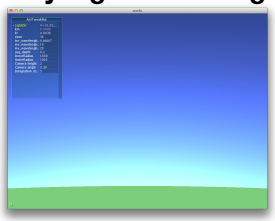


Essentially density of particles is assumed proportional to the height
The scattering constant "K" is different for {Mie, Rayleigh} scattering

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Rayleigh v.s. Mie Scattering

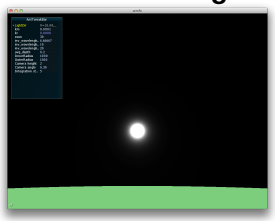
Rayleigh Scattering



Scattering of small molecules in the air
Scatters more short wavelengths

Effect: blue daylight sky
Effect: red sunset

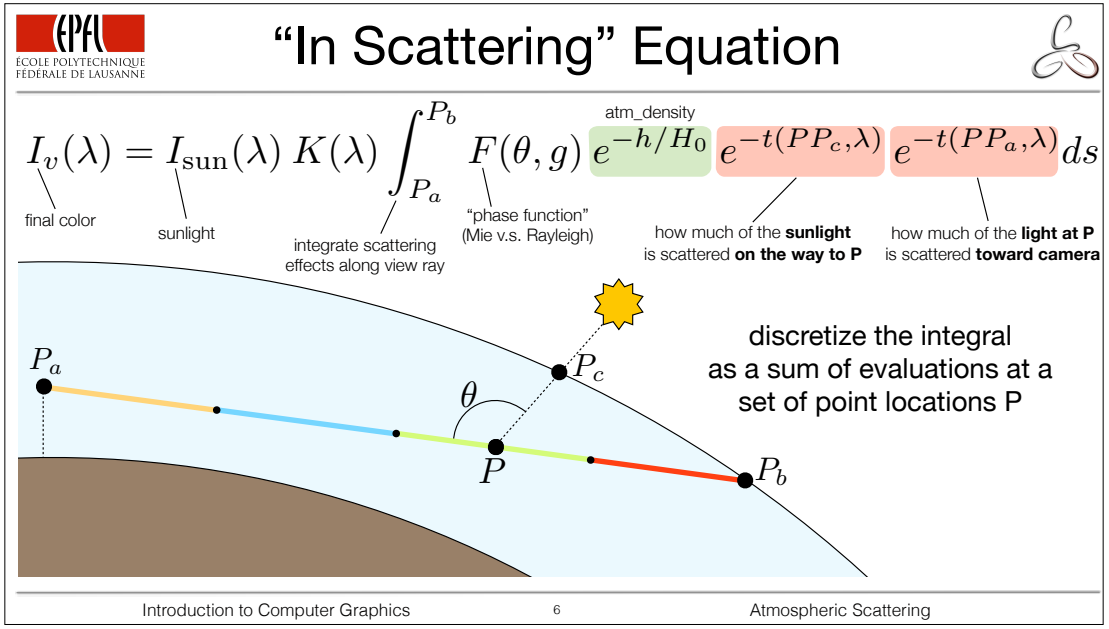
Mie Scattering



Caused by aerosol particles (dust)
Scatters equally across wavelengths

Effect: gray sky on rainy days
Effect: sun halo on cloudy days

Rayleigh (blue -> green -> red)



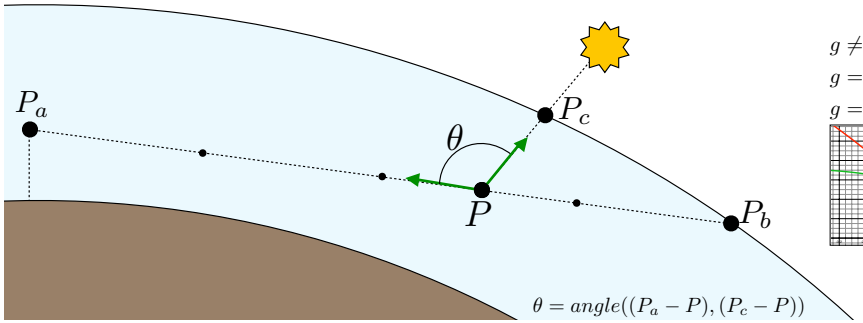
The parameter "g" of the phase function changes for {Mie, Rayleigh} (see next slide)

The Phase Function “F”

how much light is scattered in the direction of the camera (w.r.t. **angle**)

$$I_v(\lambda) = I_{\text{sun}}(\lambda)\, K(\lambda) \int_{P_a}^{P_b} F(\theta, g)\, e^{-h/H_0}\, e^{-t(PP_c,\lambda)}\, e^{-t(PP_a,\lambda)} ds$$

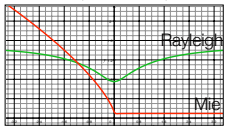
$$F(\theta, g) = \frac{3(1-g^2)}{2(2+g^2)} \frac{1+cos^2(\theta)}{(1+g^2-2g\cos(\theta))^{\frac{3}{2}}}$$



$g \neq \{-1, +1\}$

$g = [-.99, -.75]$ (Mie)

$g = 0$ (Rayleigh)

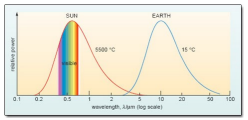


Making it Real-Time

$$I_v(\lambda) = I_{\text{sun}}(\lambda)\, K(\lambda) \int_{P_a}^{P_b} F(\theta, g)\, e^{-h/H_0}\, e^{-t(PP_c,\lambda)}\, e^{-t(PP_a,\lambda)} ds$$

Assume parallel sunlight!!!

$$I_v(\lambda) = I_{\text{sun}}(\lambda)\, K(\lambda)\, F(\theta, g)\, e^{-t(PP_c,\lambda)} \int_{P_a}^{P_b} e^{-h/H_0}\, e^{-t(PP_a,\lambda)} ds$$



Assuming (1,1,1) is ok! :)

almost a constant! ;)
(precompute and store in lookup table)

Inner integral :(

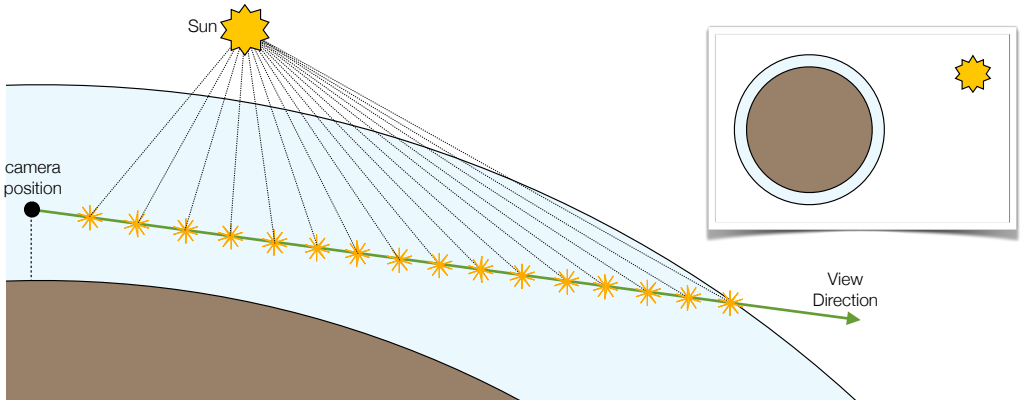
[Nishita et al. '93]

The evaluation of the outer integral is "ok" (...necessary).
But can we do something about the inner integral?
Otherwise the cost is 300 computations per-fragment
(and that just discretizing integral with 5 samples!).

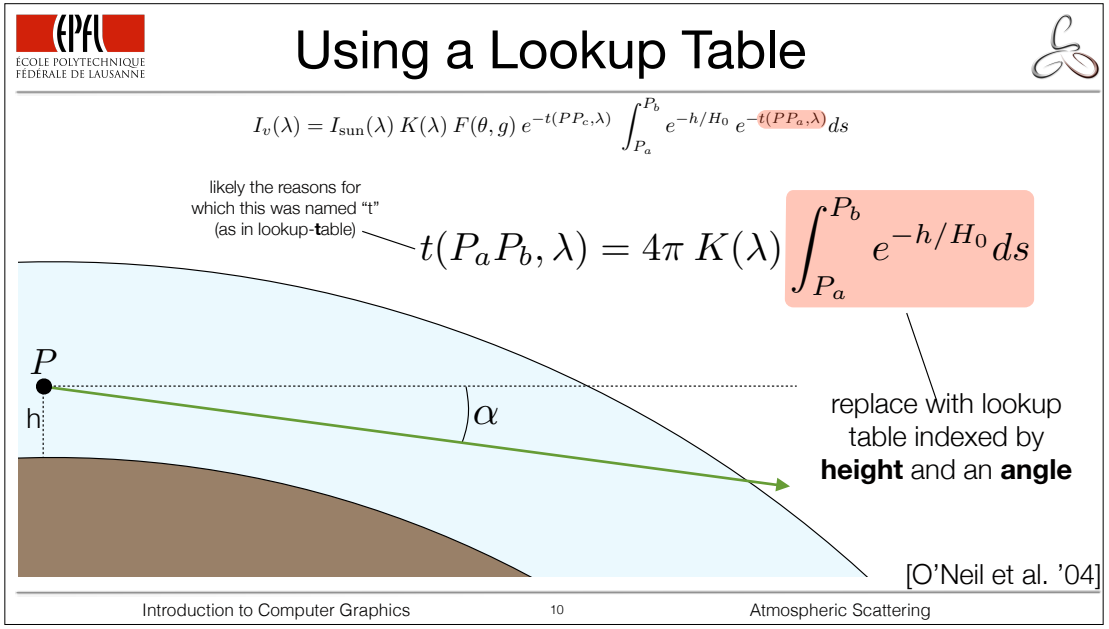
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Implementation task

Start implementing the “slow” version (aka what you have seen in the previous slide).
We provide a **simple C++ framework** to get you started on this task.



You’ll have to setup the camera, compute ray-sphere intersections, shade the terrain, etc...

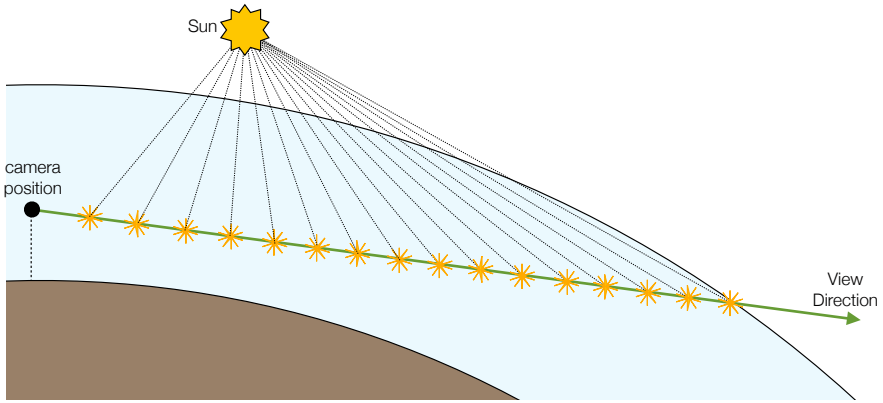


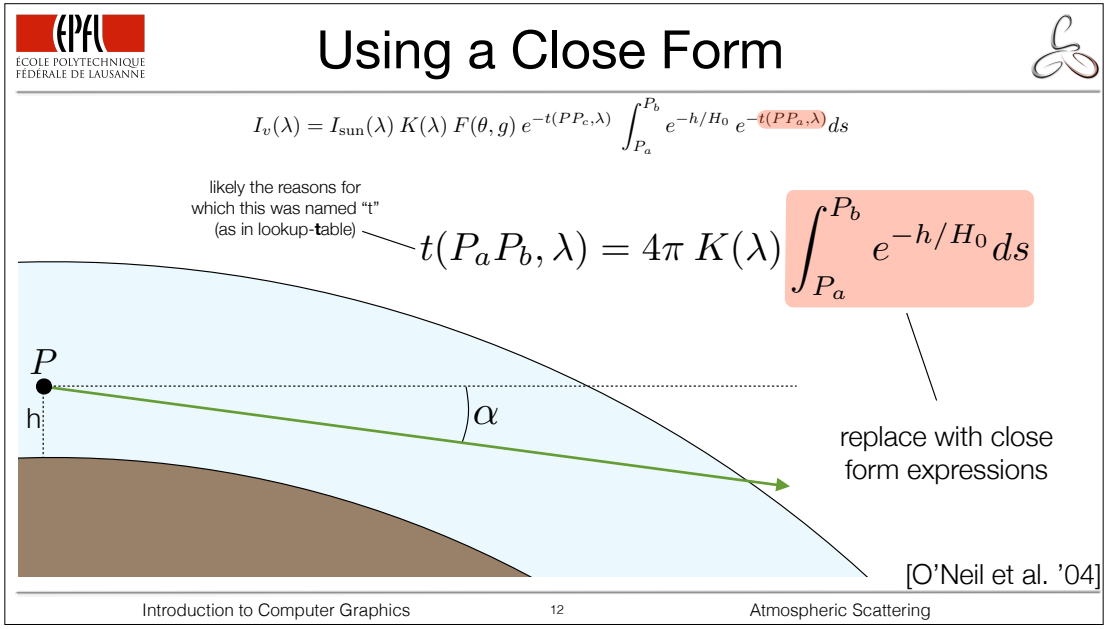
Read _VERY_ carefully the 2nd part of Section 16.3.
You have to make a few important tweaks to use the lookup table

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Implementation task (optional)

Perform the scattering integration using a lookup table.
(optional because you might want to try the close form by O'Neil directly)





Read _VERY_ carefully section 16.4.1 of the tutorial

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Final Result