High-Performance Rendering of Realistic Cumulus Clouds Using Pre-Computed Lighting

Egor Yusov

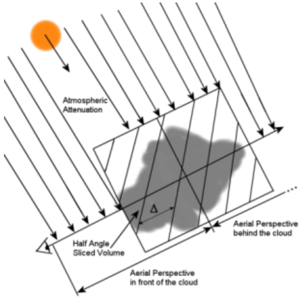
Introduction

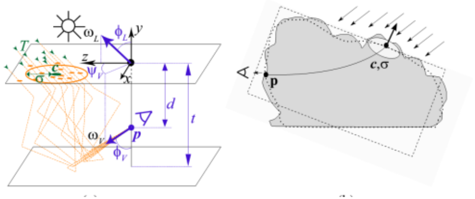
Clouds are integral part of outdoor scenes

Rendering good-looking *and* fast clouds is challenging

Existing methods

- Billboards
  - [Dobashi et al. 2000, Harris & Lastra 2001, Harris 2003, Wang 2004]
- Volume rendering (Slicing)
  - [Schpok et al. 2003, Kniss et al. 2004, Miyazaki et al. 2004, Riley et al. 2004]
- Precomputed solutions
  - [Sloan et al. 2002, Bouthors et al. 2006, Bouthors et al. 2008, Ament et al. 2013]


Atmospheric Attenuation
Δ
Half Angle Sliced Volume
Aerial Perspective behind the cloud
Aerial Perspective in front of the cloud

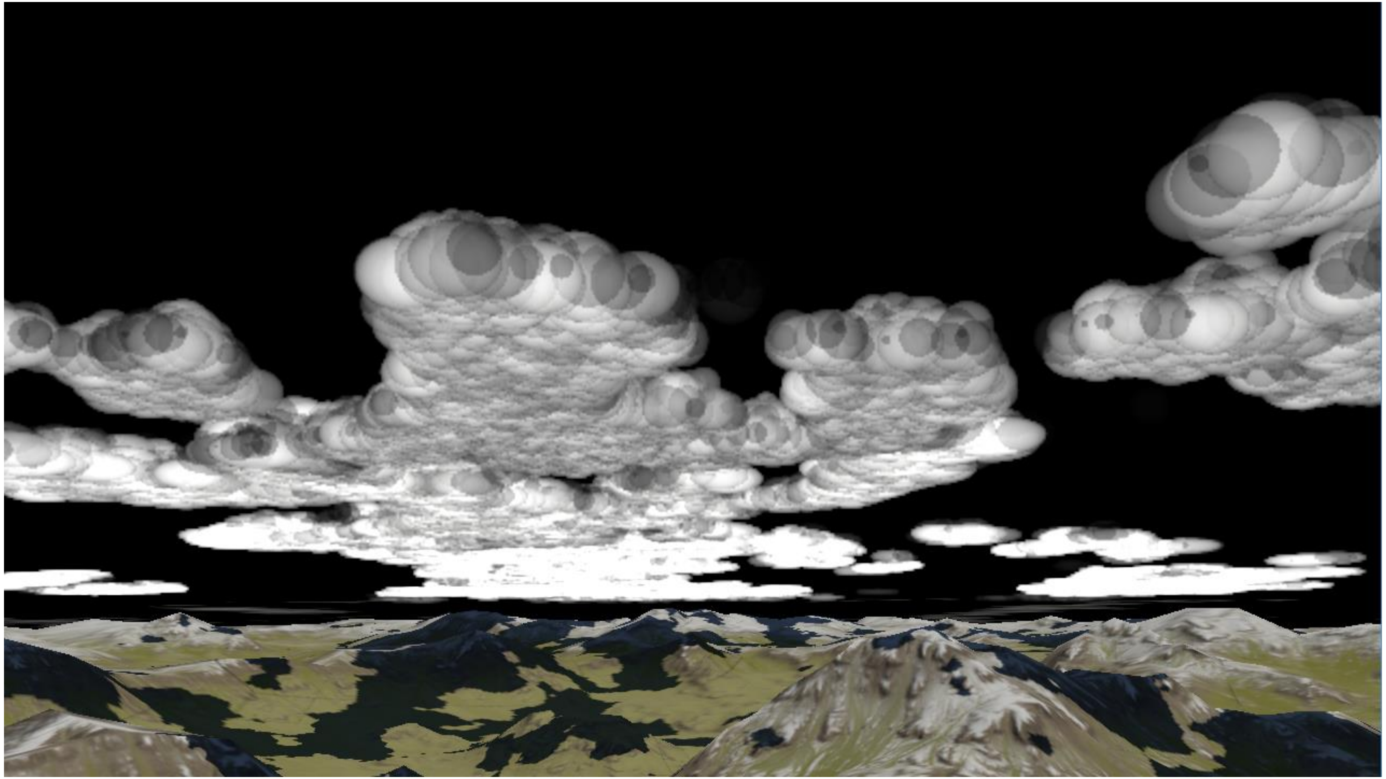


Our method

- Attempts to combine flexibility of the particle-based approaches with the quality of pre-computed techniques
- Key ideas:
  - Use volumetric particles representing the actual 3D-shapes
  - Use physically-based lighting
  - Pre-compute lighting and other quantities to avoid expensive ray-marching or slicing at run time
  - Perform volume-aware blending instead of alpha blending

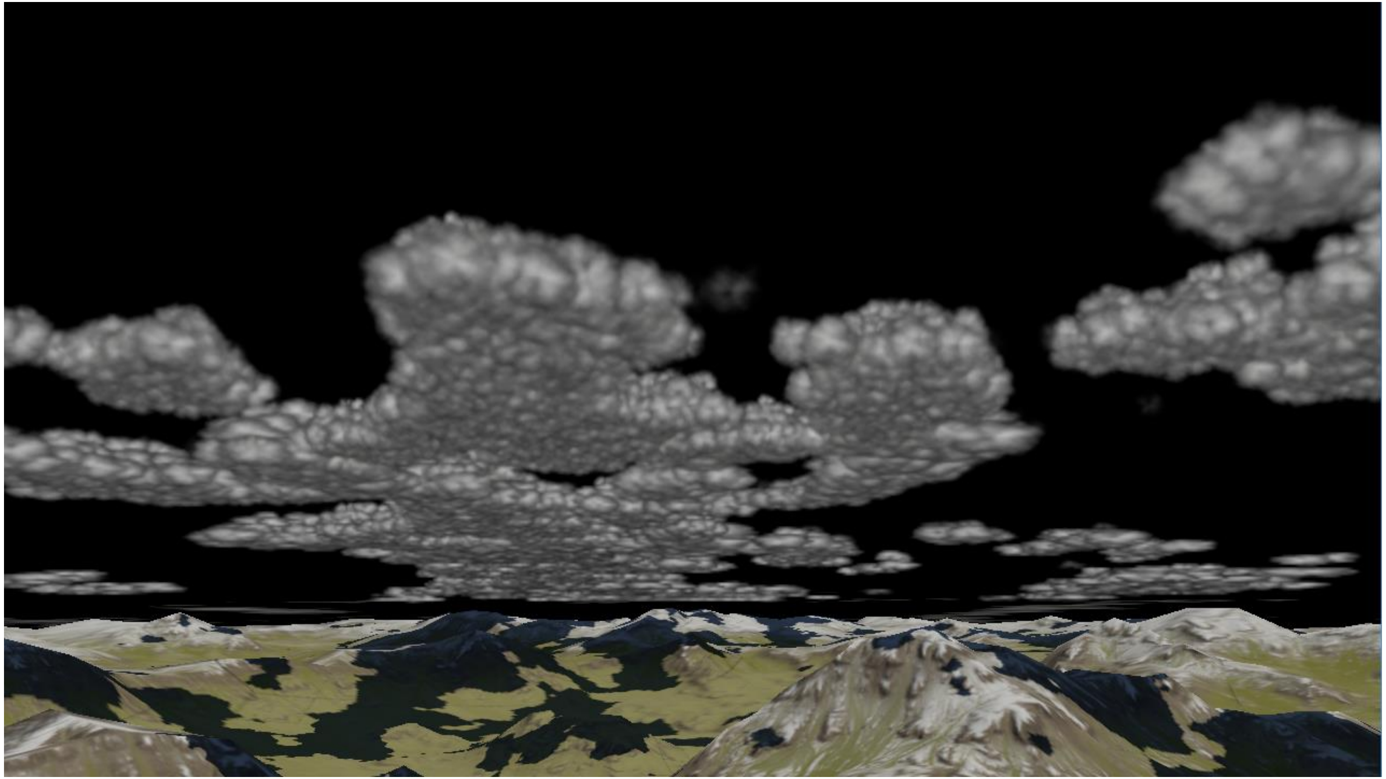
Algorithm overview

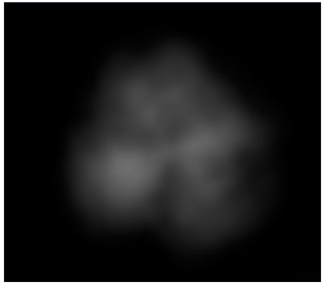
Initial step – modeling clouds with spherical particles



Algorithm overview

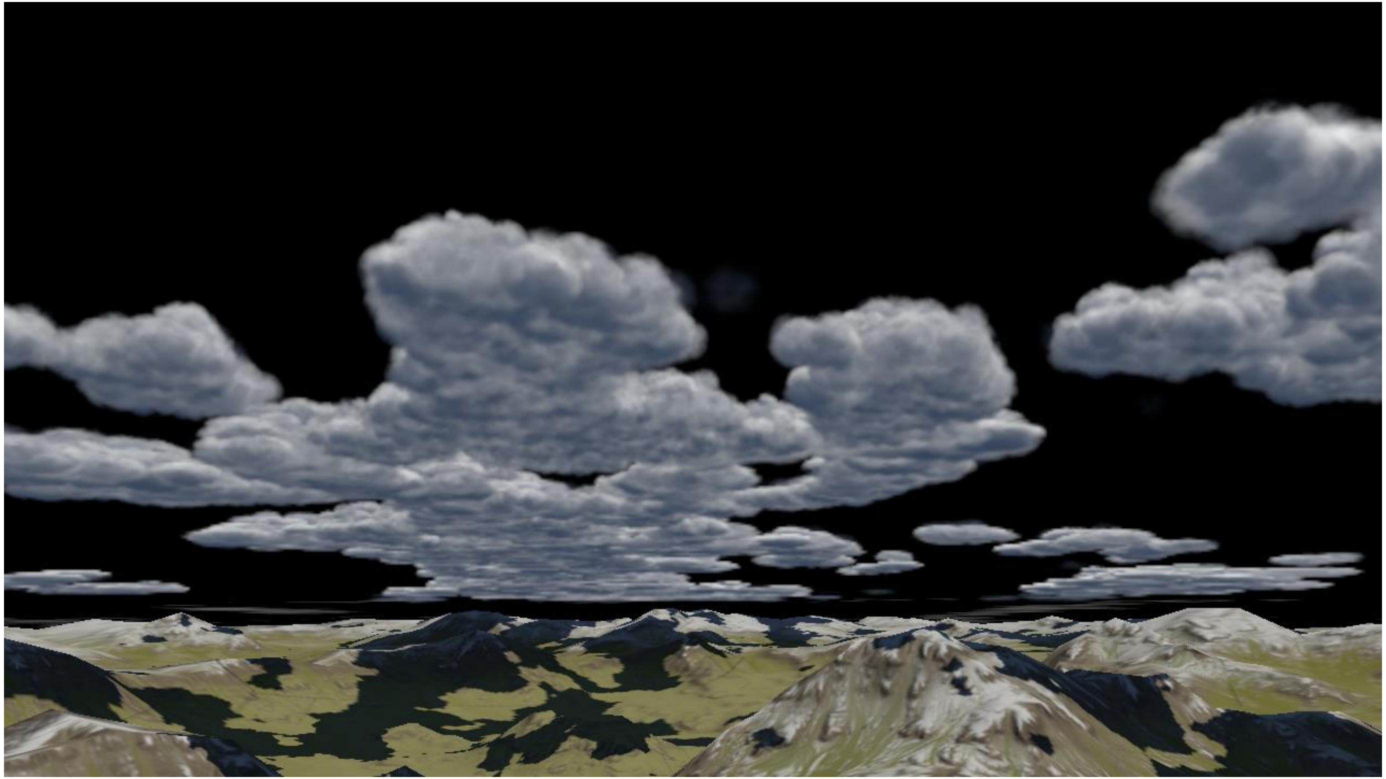
Add pre-computed cloud density and transparency

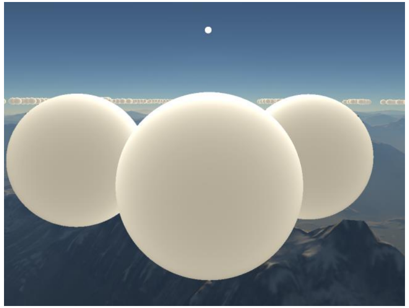




Algorithm overview

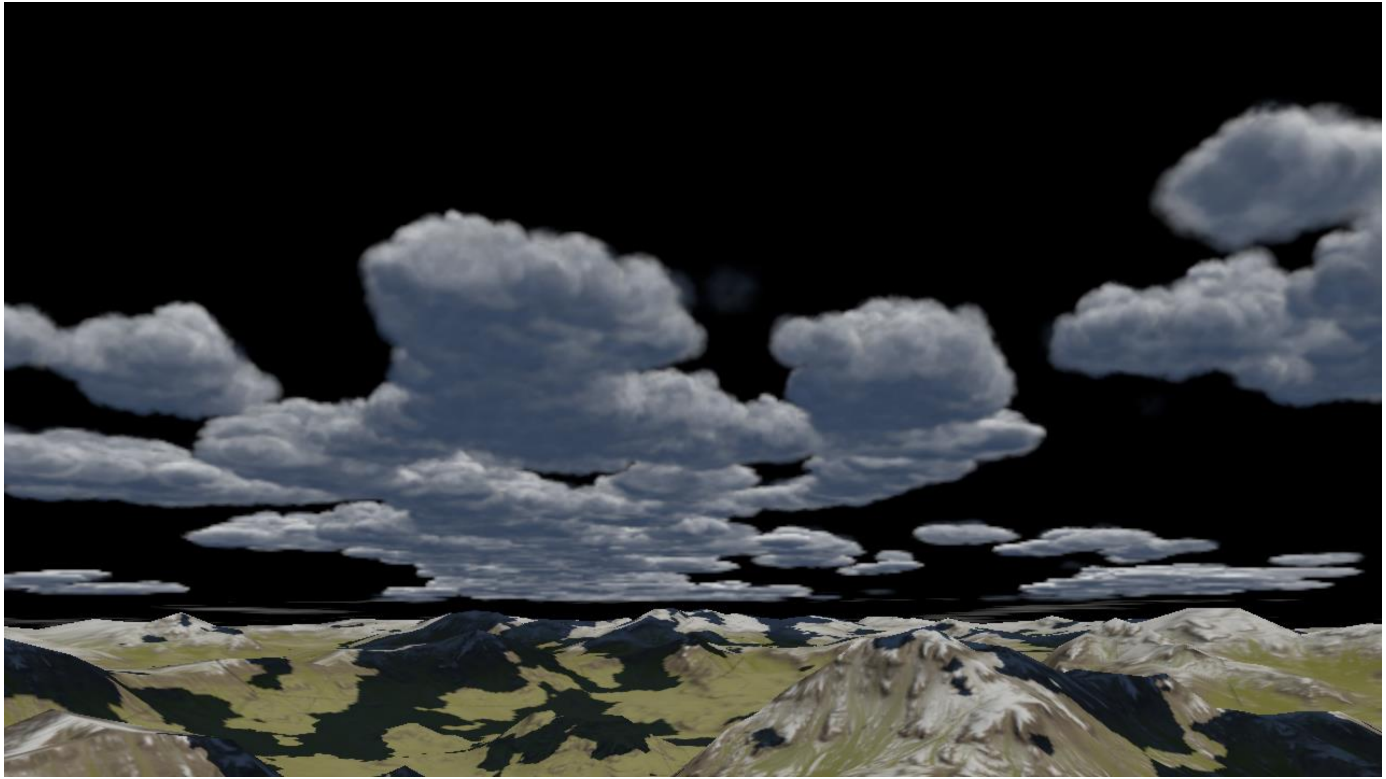
Add pre-computed light scattering





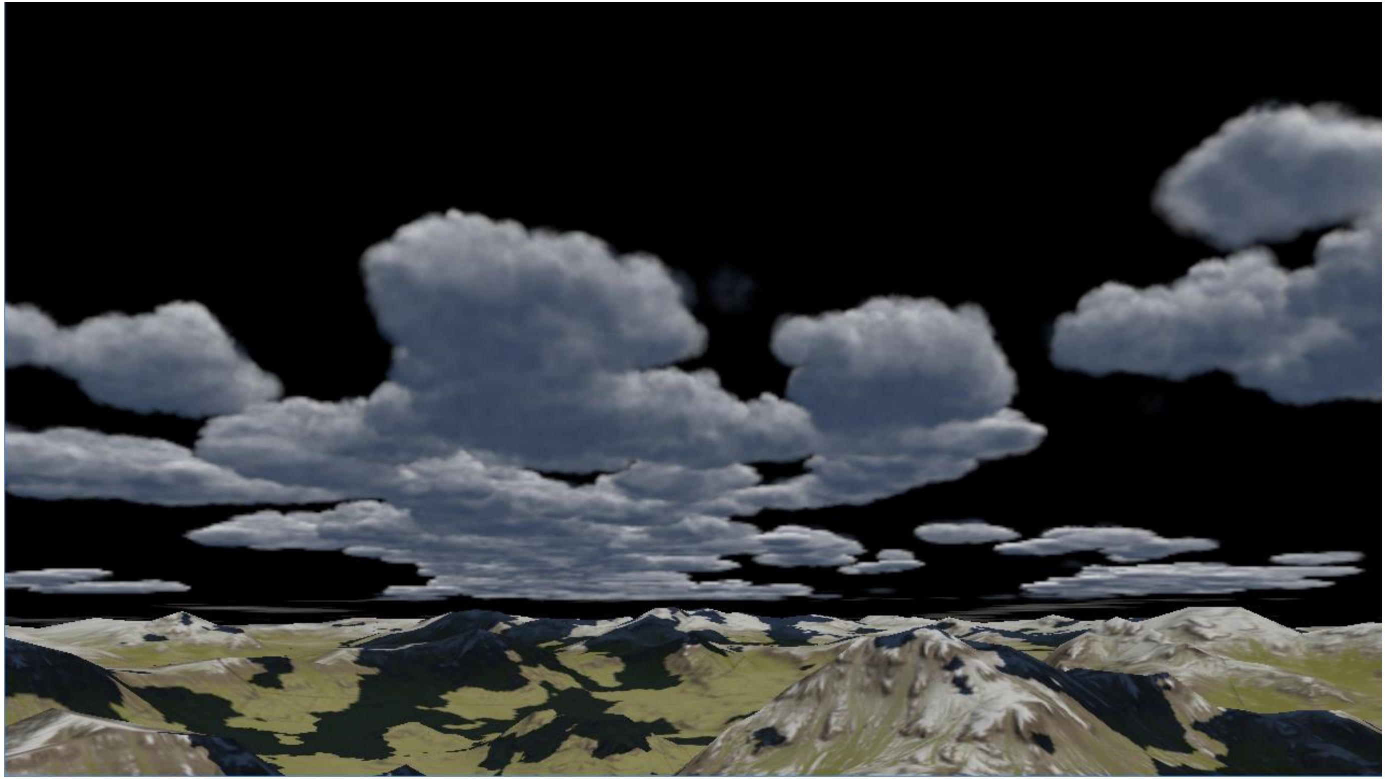
Algorithm overview

Add light occlusion



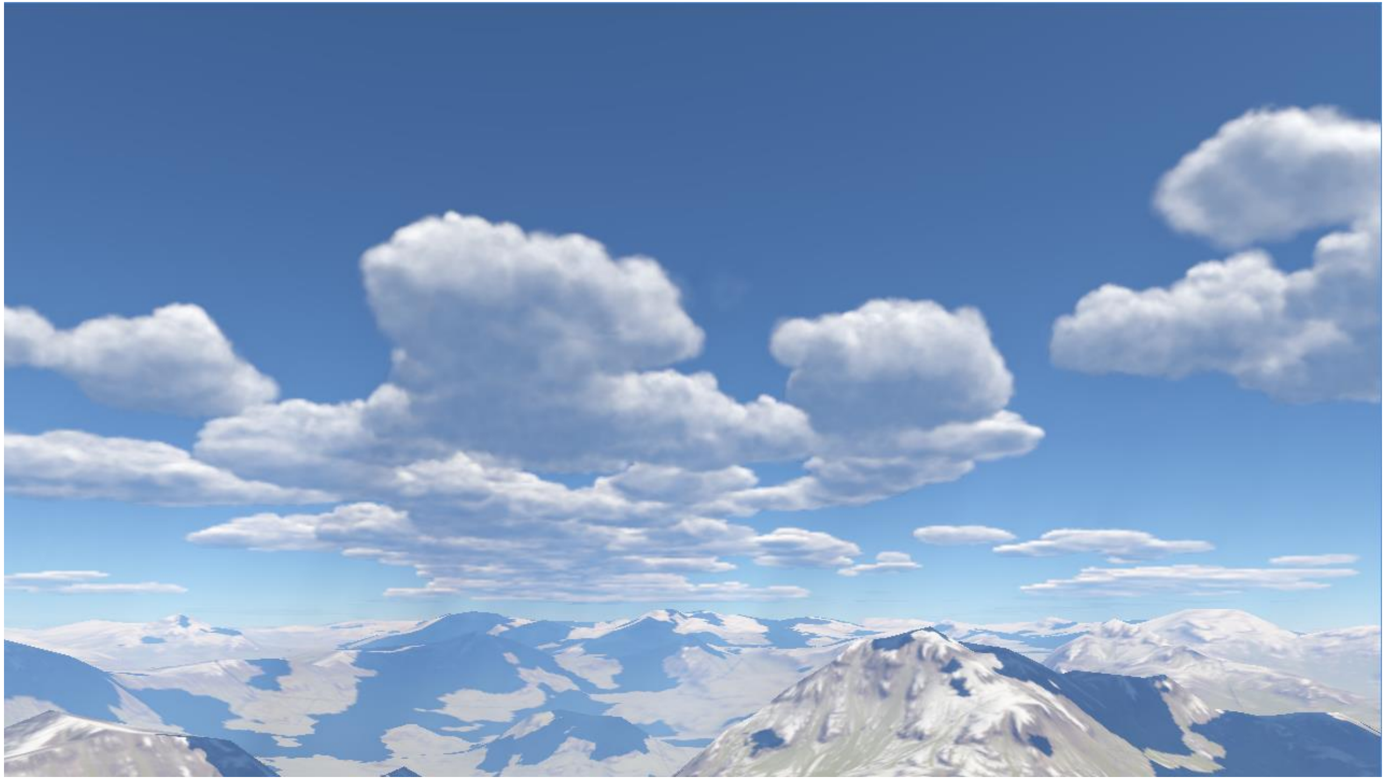
Algorithm overview

Add volume-aware blending (enabled by Pixel Sync)

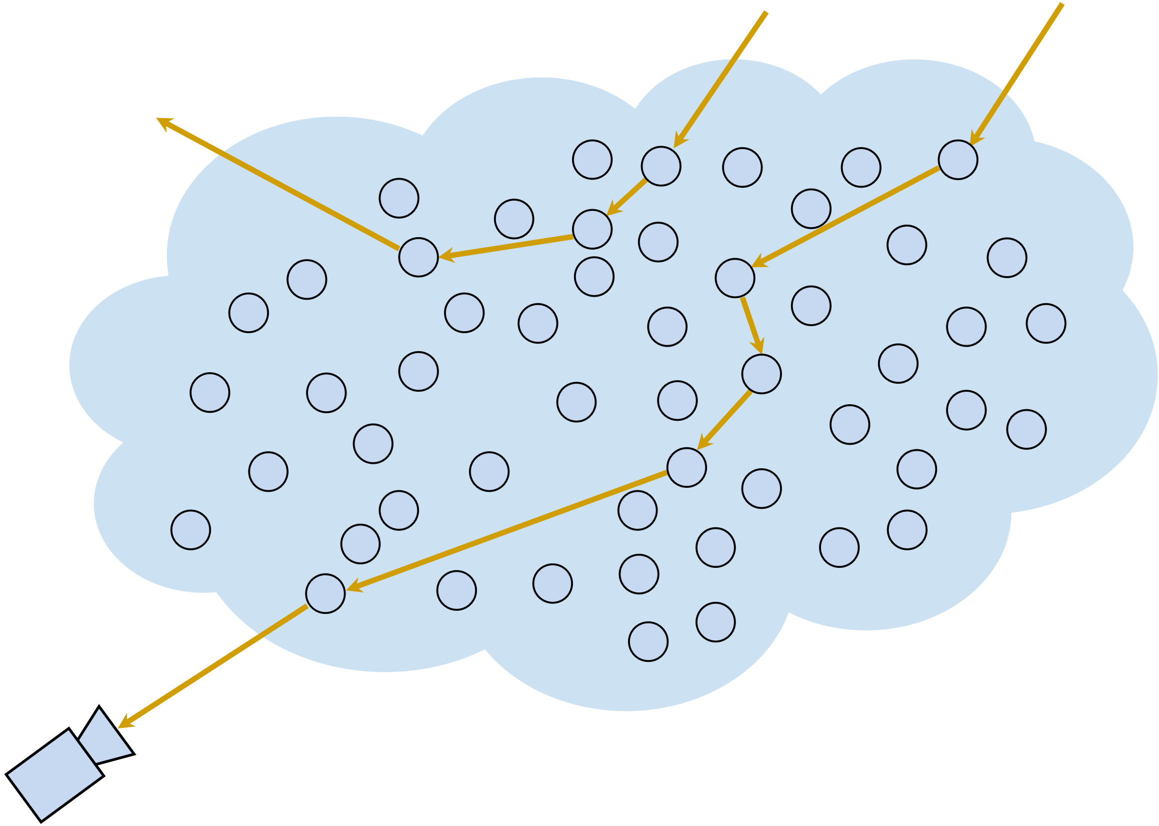


Algorithm overview

Add light scattering



Scattering physics



Scattering physics

Optical depth integral

Light gets attenuated while it travels through the cloud

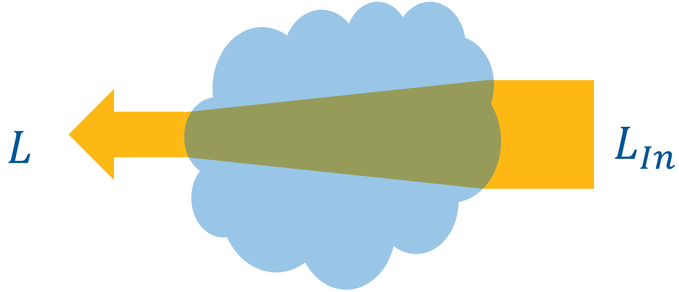
No absorption => only out-scattering attenuates the light

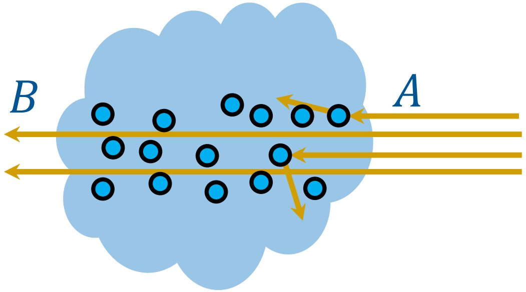
Optical depth is the amount of scattering matter on the way of light:

$$T(\mathbf{A} \to \mathbf{B}) = \int_{\mathbf{A}}^{\mathbf{B}} \beta(\mathbf{P})\, ds$$

Transmittance is the fraction of light survived out-scattering:

$$L = e^{-T(\mathbf{A} \to \mathbf{B})} \cdot L_{In}$$





Scattering physics

Single-scattering integral:

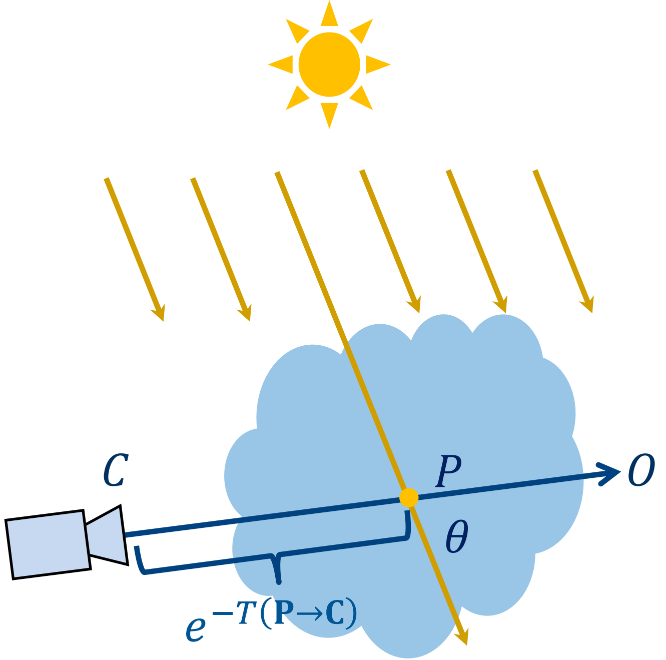
$$L_{In} = p(\theta)\int_{\mathbf{C}}^{\mathbf{O}} e^{-T(\mathbf{P}\to\mathbf{C})}\,\beta(\mathbf{P})\,L(\mathbf{P})\;ds$$

$L(\mathbf{P})$ is the light intensity at point P

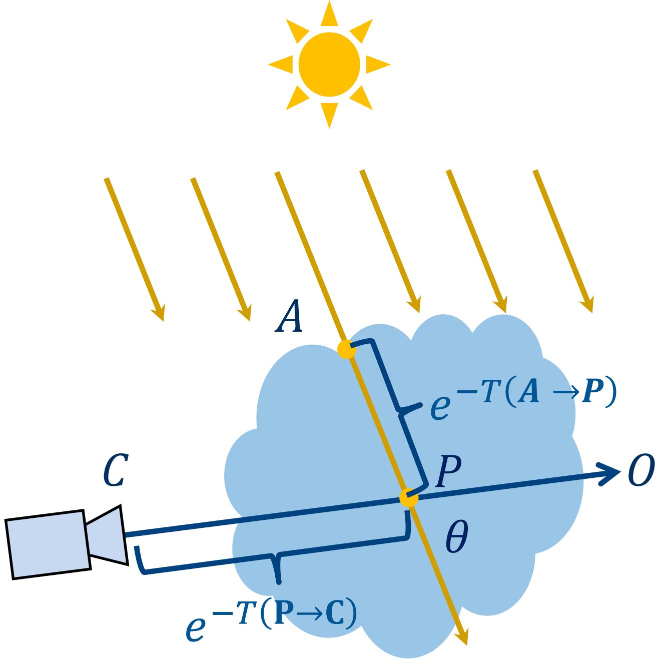
$\beta(\mathbf{P})$ is the scattering coefficient at point P

$T(\mathbf{P} \to \mathbf{C})$ is the optical thickness of the media between points P and C

$p(\theta)$ is the phase function



Scattering physics



A
$e^{-T(\boldsymbol{A} \rightarrow \boldsymbol{P})}$
C
P
O
θ
$e^{-T(\mathbf{P}\rightarrow\mathbf{C})}$

Light is also attenuated in the cloud before it reaches the scattering point:

$$L(\mathbf{P}) = L\ e^{-T(\boldsymbol{A} \rightarrow \boldsymbol{P})}$$

L is the light intensity outside the cloud

Let’s now look at our integral:

$$\int_{\mathbf{P}}^{\mathbf{C}} \beta(\mathbf{P})\, ds \qquad \int_{\mathbf{A}}^{\mathbf{P}} \beta(\mathbf{P})\, ds$$

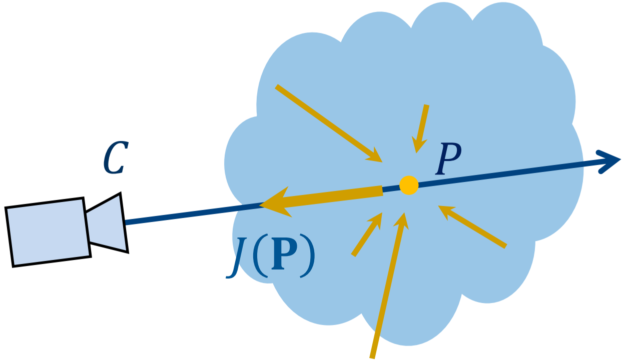
$$L_{In} = p(\theta) \int_{\mathbf{C}}^{\mathbf{O}} e^{-T(\mathbf{P}\rightarrow\mathbf{C})} \beta(\mathbf{P}) L\ e^{-T(\boldsymbol{A} \rightarrow \boldsymbol{P})}\, ds$$

Scattering physics

Multiple scattering

$$L \quad = p(\theta) \int_{\mathbf{C}}^{\mathbf{O}} e^{-T(\mathbf{P}\rightarrow\mathbf{C})}\, \beta(\mathbf{P})\, J(\mathbf{P})\; ds$$

$$J(\mathbf{P}) = \int_{\mathbf{\Omega}} L\,(\omega) p(\theta) d\omega$$

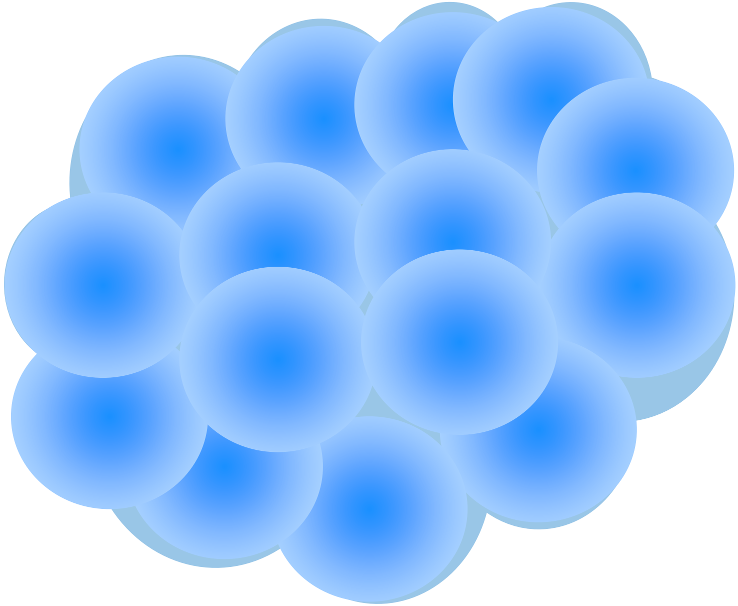


$\mathbf{\Omega}$ is the whole set of directions

Pre-computed lighting

The main idea is to

- Precompute physically-based lighting for simple shapes
- Construct clouds from these simple shapes
- The term **Particle** will now refer to these basic shapes (not individual tiny droplets)



Pre-computing optical depth

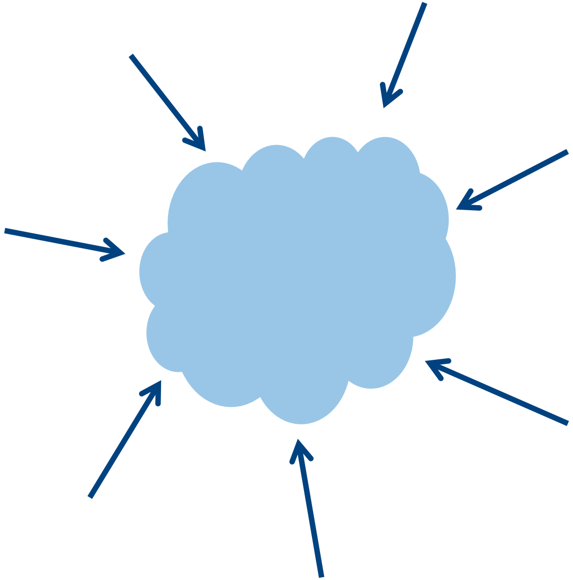
$$T(\mathbf{A} \to \mathbf{B}) = \int_{\mathbf{A}}^{\mathbf{B}} \beta(\mathbf{P})\, ds$$

Typical way to evaluate optical depth is ray marching

- Impractical to do in real-time

For a known density distribution, the integral can be evaluated once and stored in a look-up table for all possible viewpoints and directions

- No ray marching at run-time
- Fast evaluation for the price of memory

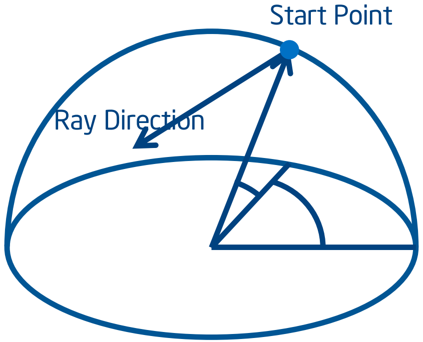


Pre-computing optical depth

$$T(\mathbf{A} \to \mathbf{B}) = \int_{\mathbf{A}}^{\mathbf{B}} \beta(\mathbf{P})\, ds$$

Parameterization

- We need to describe all start points on the sphere and all directions
- Two angles describe start point on the sphere
- Two angles describe view direction
- 4D look-up table is required

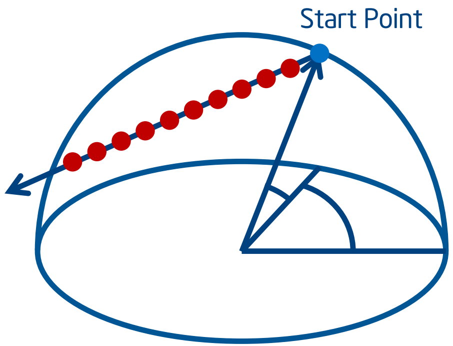


Pre-computing optical depth

$$T(\mathbf{A} \to \mathbf{B}) = \int_{\mathbf{A}}^{\mathbf{B}} \beta(\mathbf{P})\, ds$$

Integration

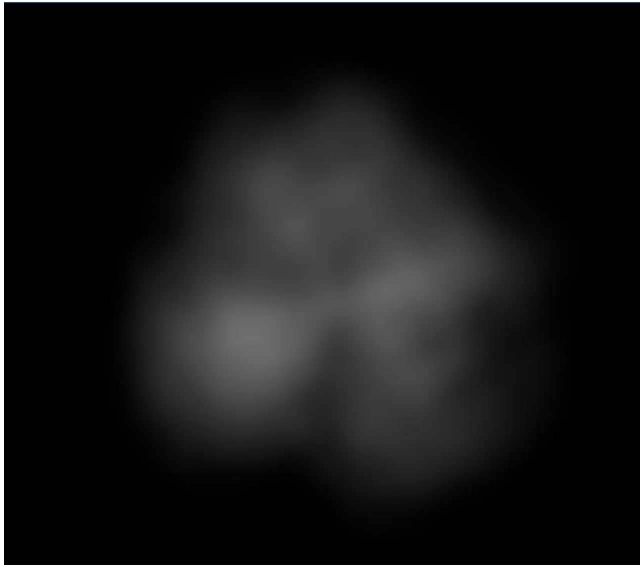
- Integration is performed by stepping along the ray and numerically computing optical thickness
  - Cloud density at each step is determined through 3D noise
- 4D look-up table is implemented as 3D texture
  - For look-up, manual filtering across 4th coordinate is necessary



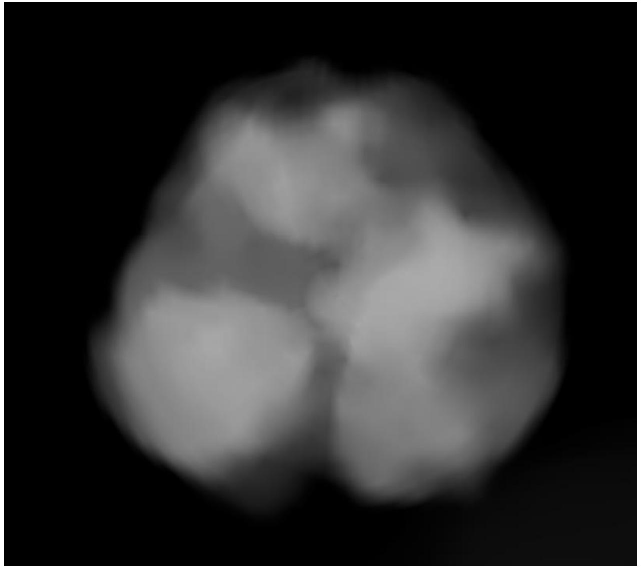
Pre-computing optical depth

3D Noise generation

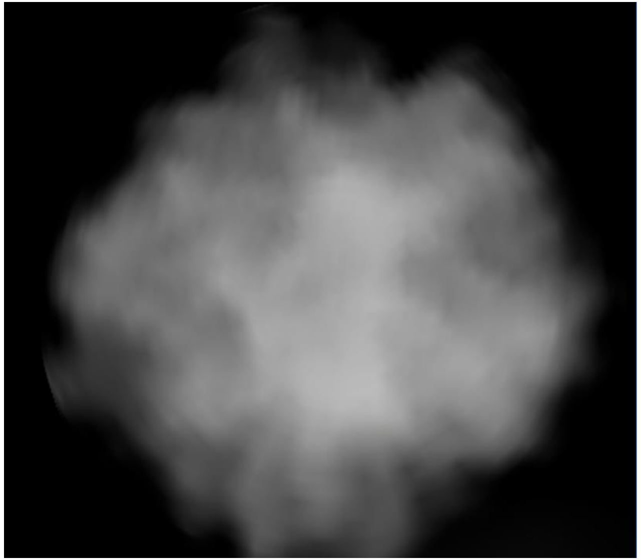
Radial falloff+3D noise



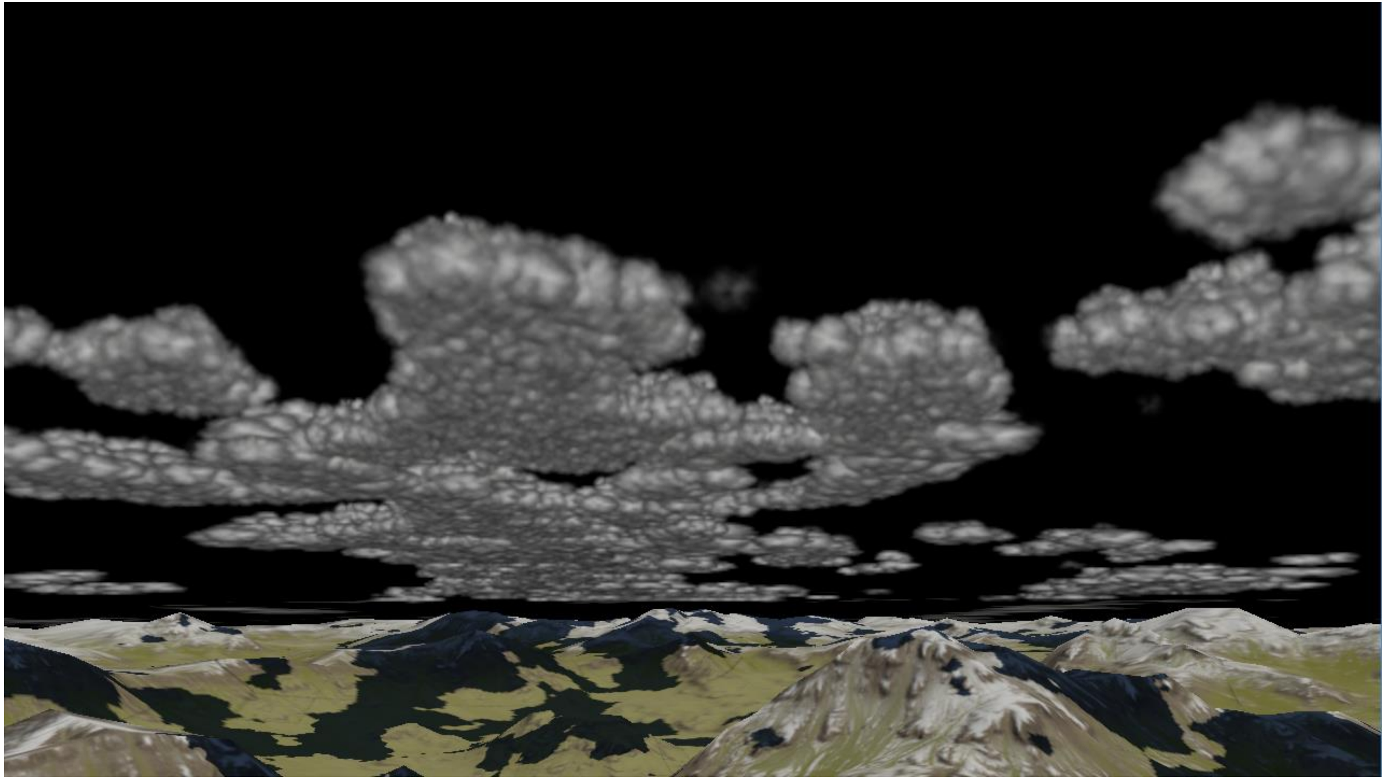
Thresholding



Pyroclastic style



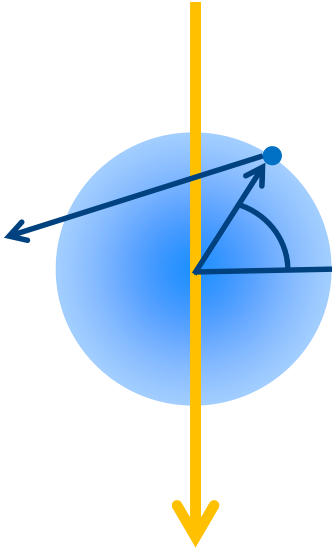
Pre-computing optical depth



Pre-computing scattering

- Let's consider spherically symmetrical particle
- Any start point on the sphere can be described by a single angle
- View direction is described by two angles
- Thus 3 parameters are necessary to describe any start point and view direction -> 3D look-up table

$$L = \int_{\mathbf{C}}^{\mathbf{O}} e^{-T(\mathbf{P}\to\mathbf{C})}\,\beta(\mathbf{P})\left(\int_{\boldsymbol{\Omega}} L\,p(\theta)d\omega\right)ds$$



Pre-computing scattering

Intermediate 4D table is used to store radiance for every point in the sphere

For each scattering order:

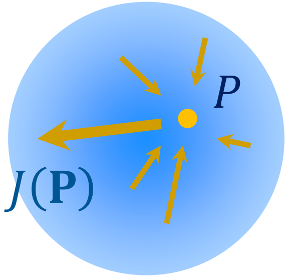
1. Compute $J(\mathbf{P})$ for every point and direction inside the sphere by integrating previous order scattering

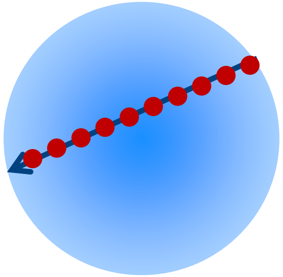
$$J_n = \int_{\Omega} L_{n-1}(\omega) p(\theta) d\omega$$

2. Compute current order inscattering by numerical integration of J_n:

$$L_n = \int_{C}^{O} e^{-T(\mathbf{P}\rightarrow\mathbf{C})} \beta(\mathbf{P}) J_n(\mathbf{P}) ds$$

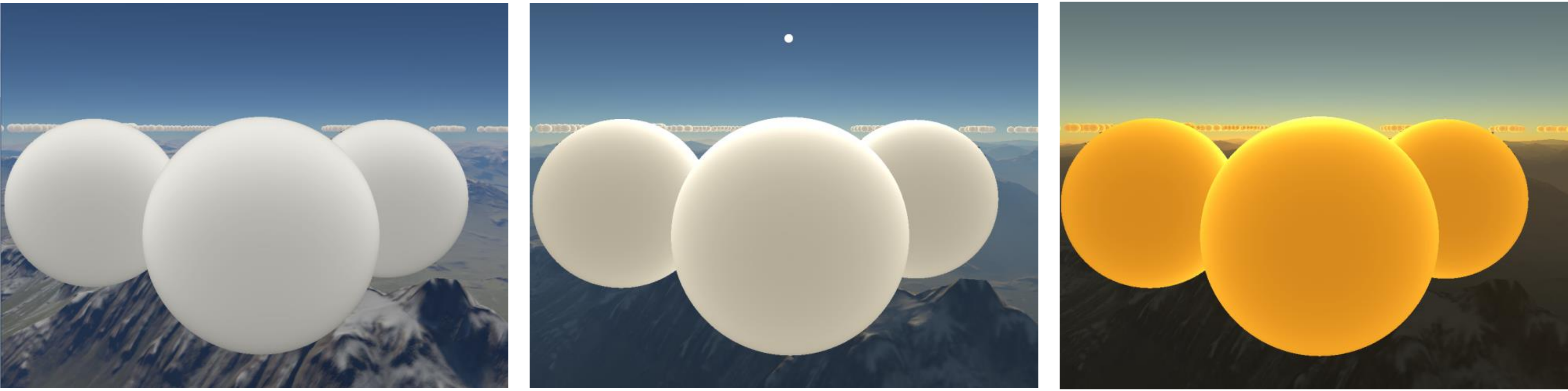
3. Add current scattering order to the total look-up table





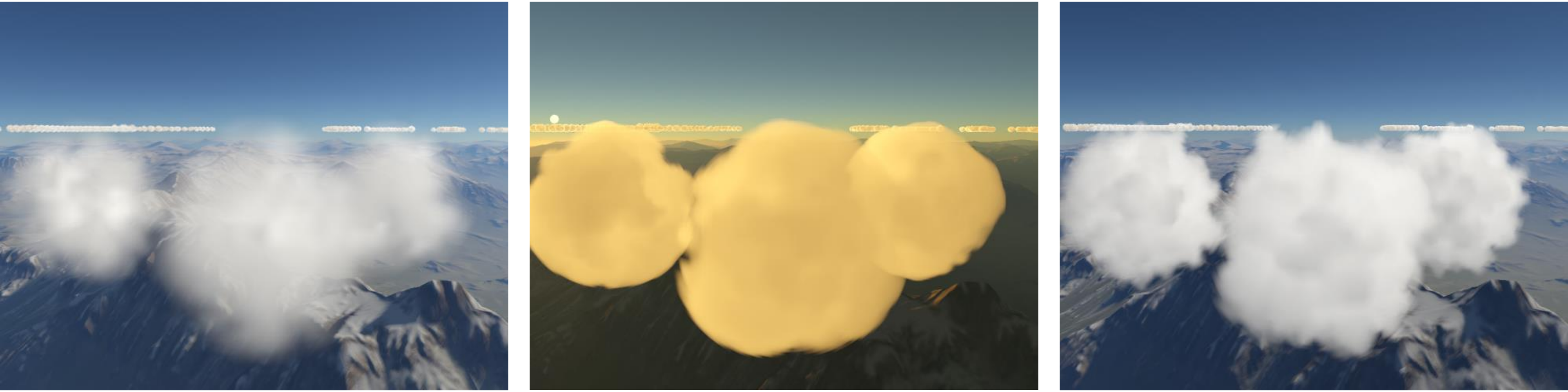
Pre-computing scattering

Pre-computed scattering for different light orientations

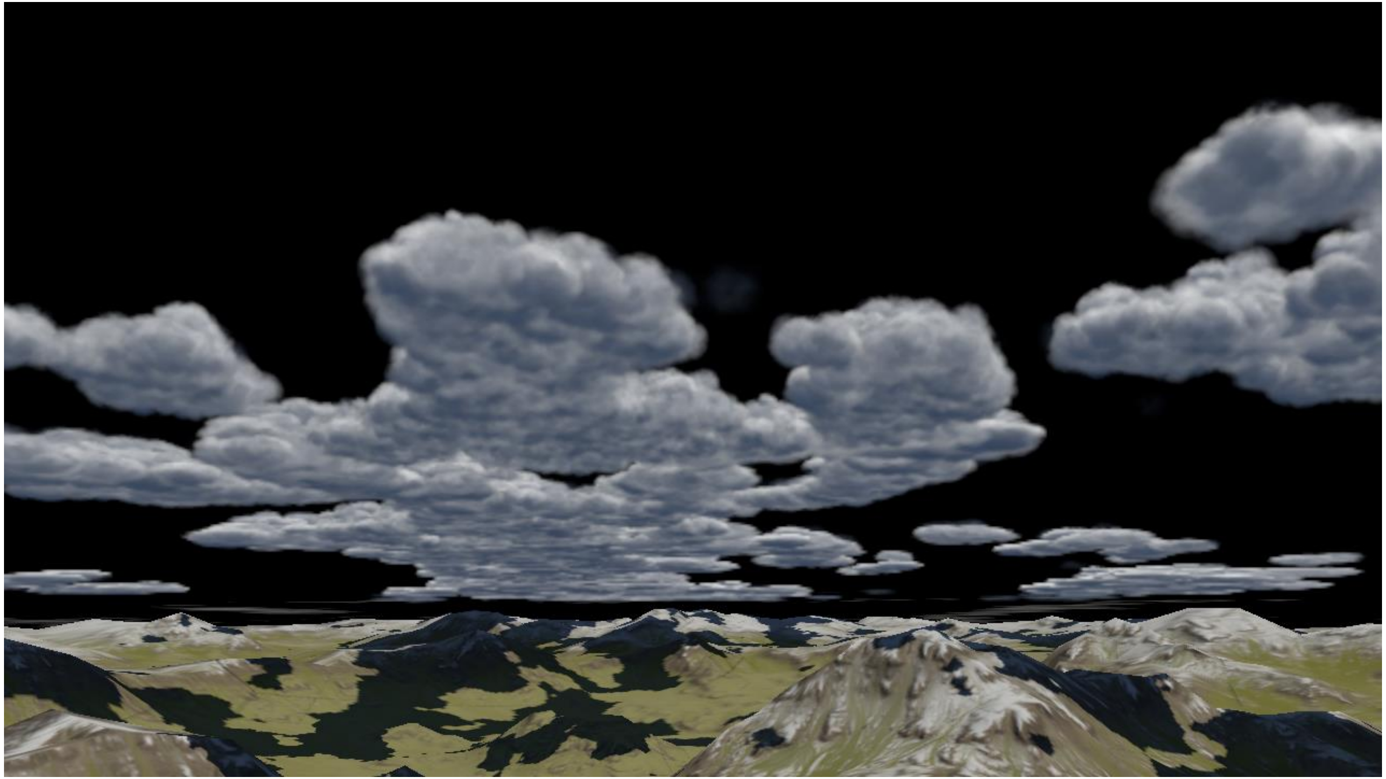


Pre-computing scattering

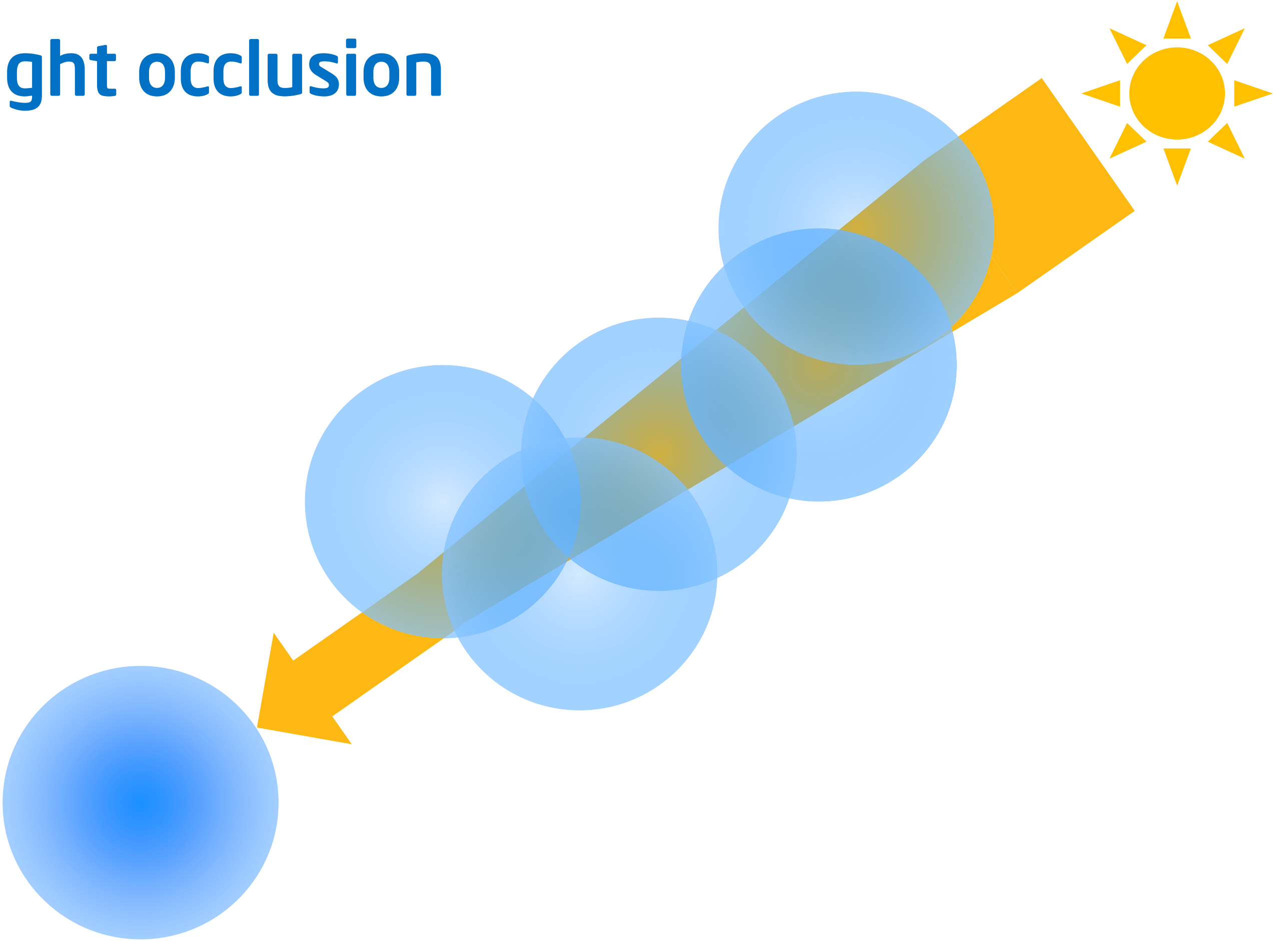
Combining pre-computed lighting and pre-computed cloud density



Pre-computing scattering



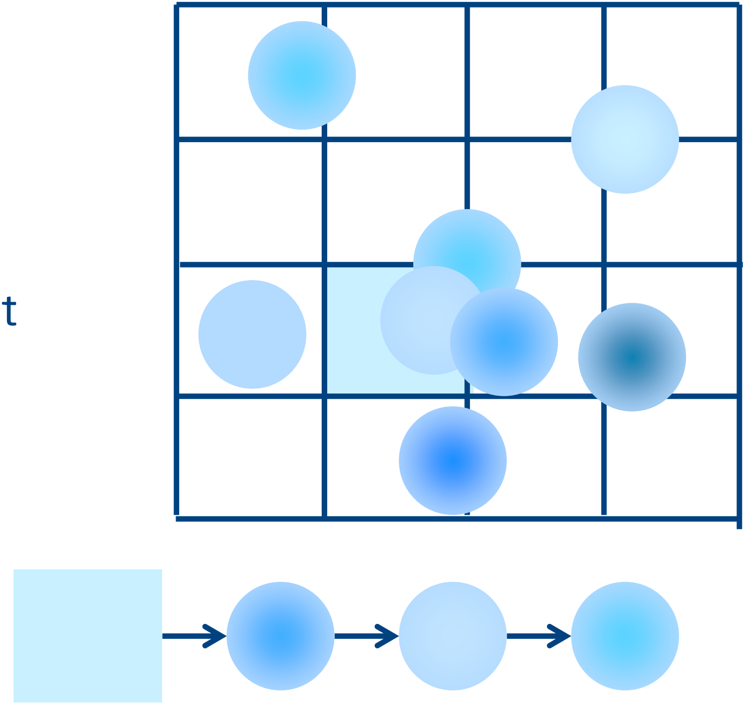
Computing light occlusion



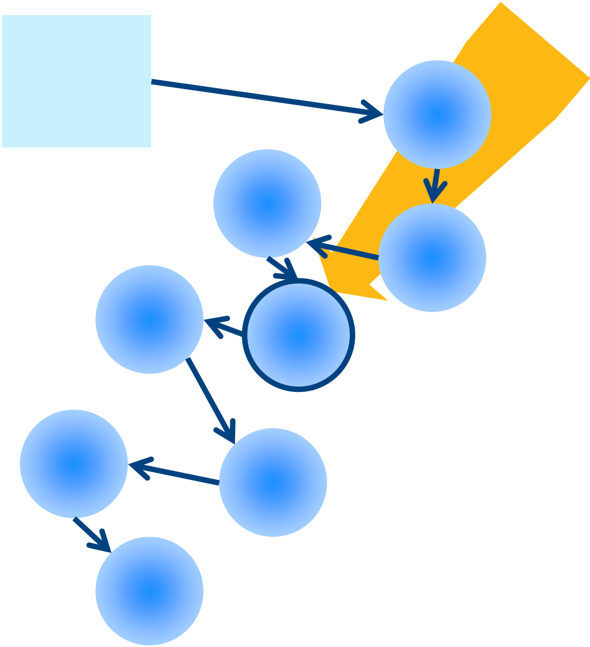
Computing light occlusion

Tiling

- The scene is rasterized from the light over the tile grid
  - One tile is one pixel
- Each particle is assigned to the tile
  - Screen-size buffer is used to store index of the first particle in the list
  - Append buffer is used to store the lists elements
- Pixel Shader Ordering is used to preserve original particle order (sorted from the light)



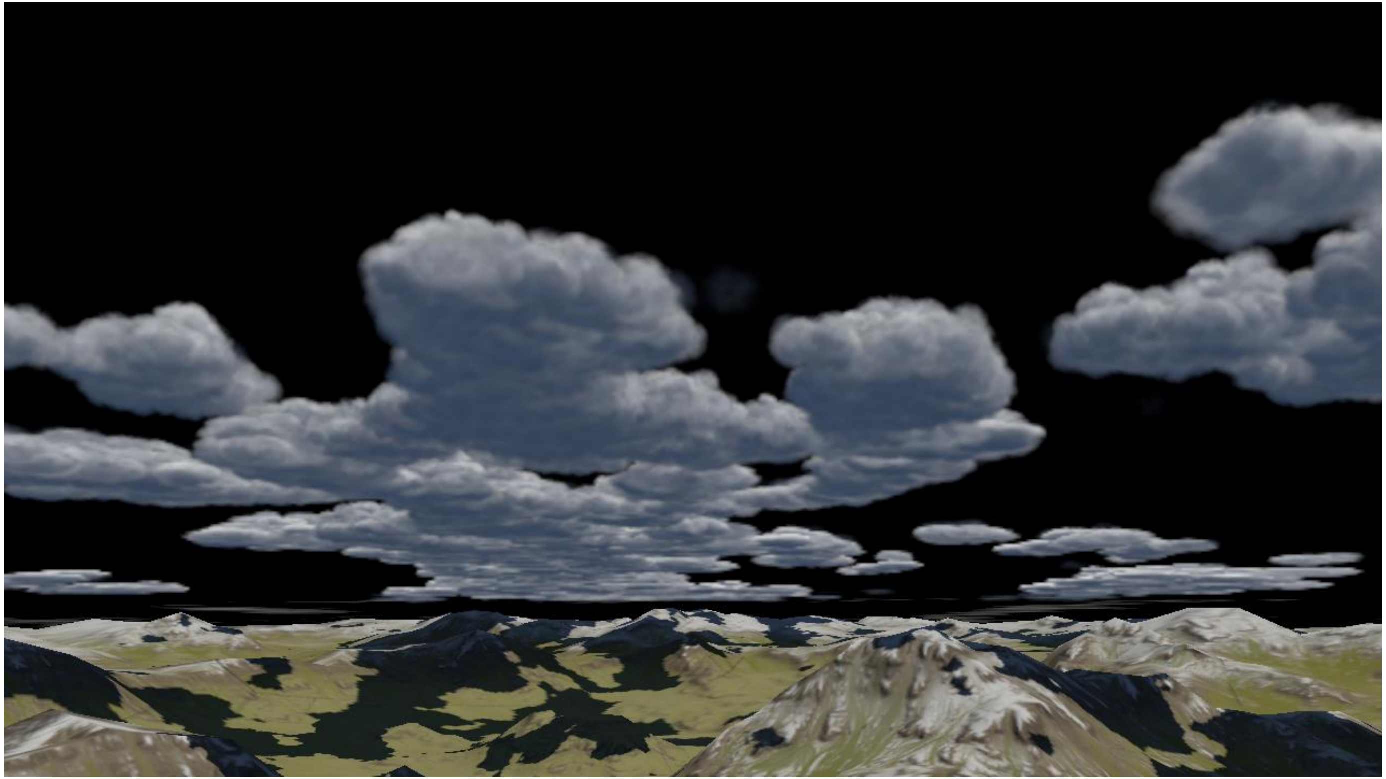
Computing light occlusion



Traversing lists

- Processing is done by the compute shader
- Each particle finds a tile it belongs to
- The shader then goes through the list of the tile and computes opacity of particles on the light path
- The loop is terminated as soon as current particle is reached
- Or if total transparency reaches threshold (0.01)

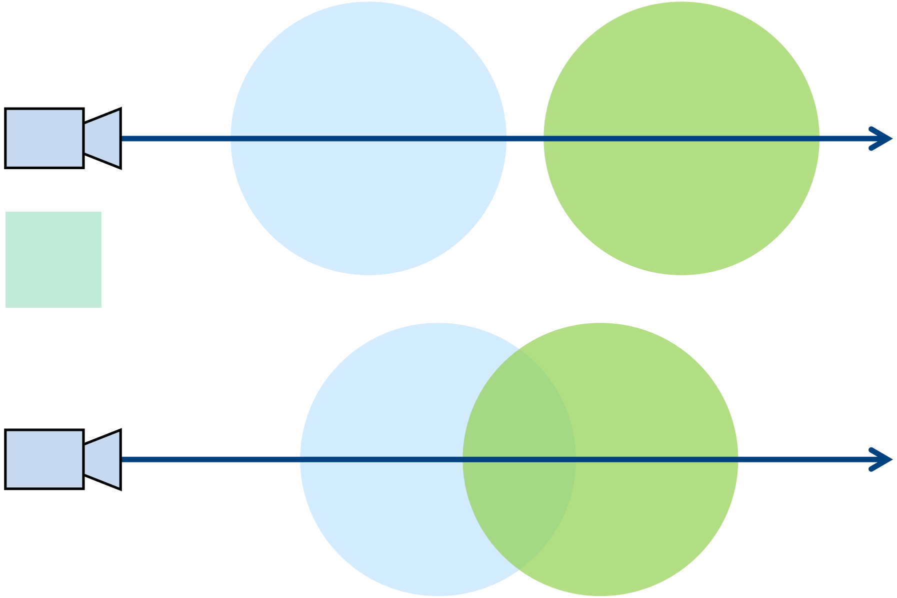
Computing light occlusion



Volume-aware blending

Blending volumetric particles

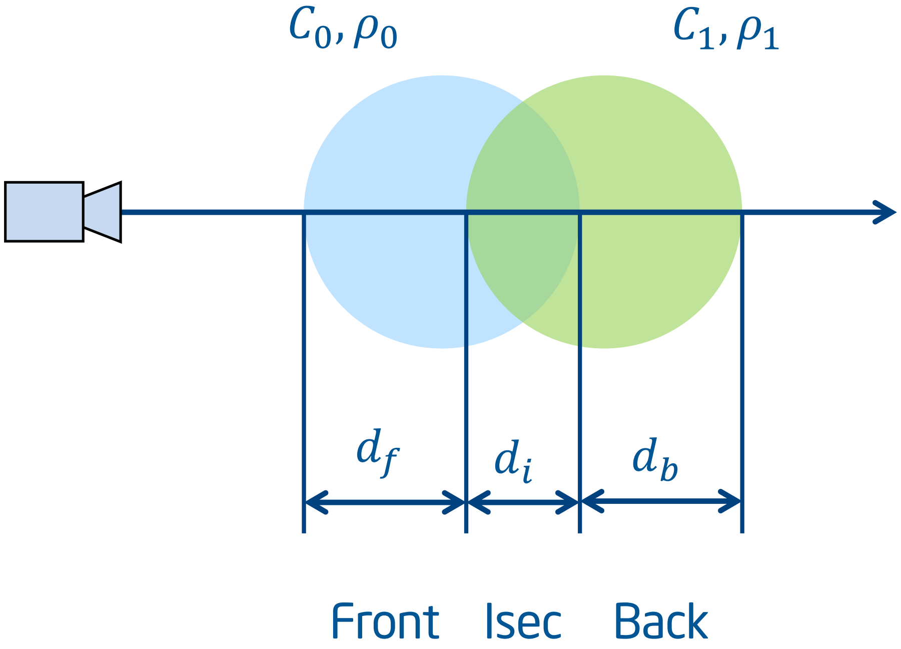
- If particles do not overlap, blending is trivial
- How can we correctly blend overlapping particles?



Volume-aware blending

Blending volumetric particles

- Suppose we have two overlapping particles with color and density C_0, ρ_0 and C_1, ρ_1
- Back:
  - $T_{Back} = e^{-\rho_1 \cdot d_b \cdot \beta}$
  - $C_{Back} = C_1 \cdot (1 - T_{Back})$
- Front:
  - $T_{Front} = e^{-\rho_0 \cdot d_f \cdot \beta}$
  - $C_{Front} = C_0 \cdot (1 - T_{Front})$
- Intersection:
  - $T_{Isec} = e^{-(\rho_0 + \rho_1) \cdot d_i \cdot \beta}$
  - $C_{Isec} = \frac{C_0 \rho_0 + C_1 \rho_1}{\rho_0 + \rho_1}(1 - T_{Isec})$



Volume-aware blending

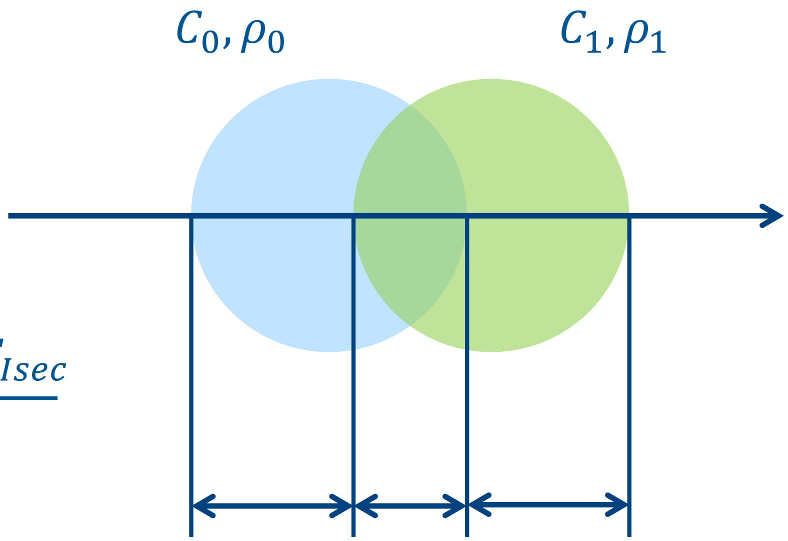
Blending volumetric particles

- Final color and transparency:

$$T_{Final} = T_{Front} \cdot T_{Isec} \cdot T_{Back}$$

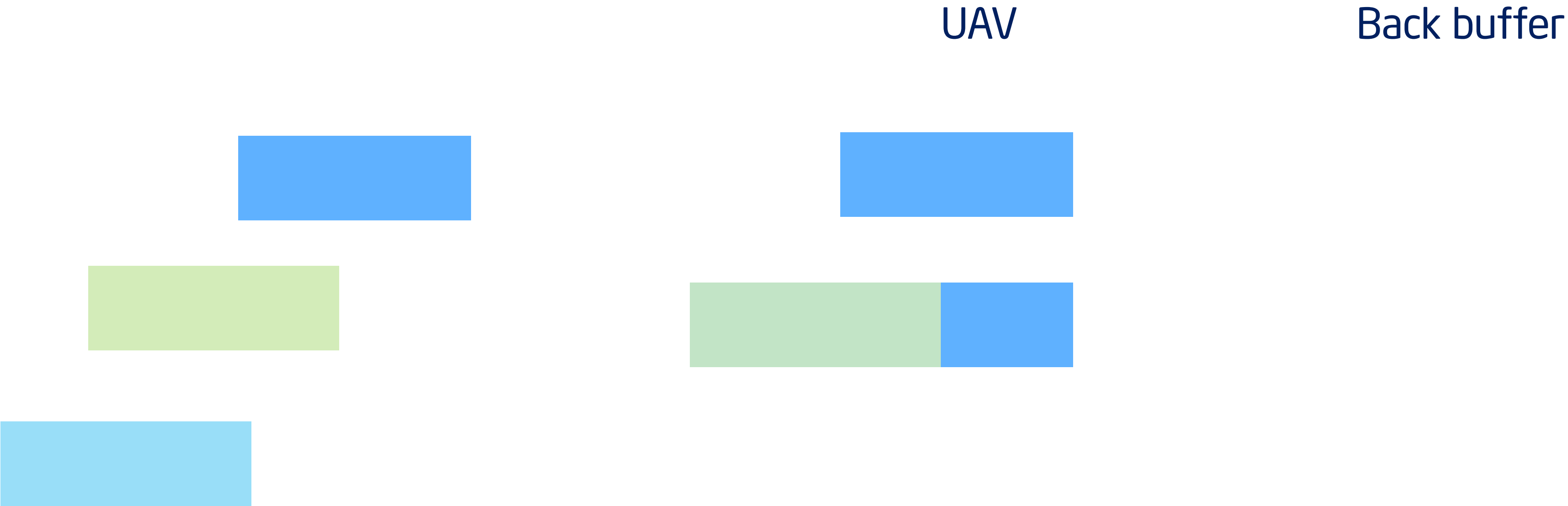
$$C_{Final} = \frac{C_{Front} + C_{Isec} \cdot T_{Front} + C_{Back} \cdot T_{Front} \cdot T_{Isec}}{1 - T_{Final}}$$

- Division by $1 - T_{Final}$ because we do not want alpha pre-multiplied color



Volume-aware blending

Blending volumetric particles - Implementation

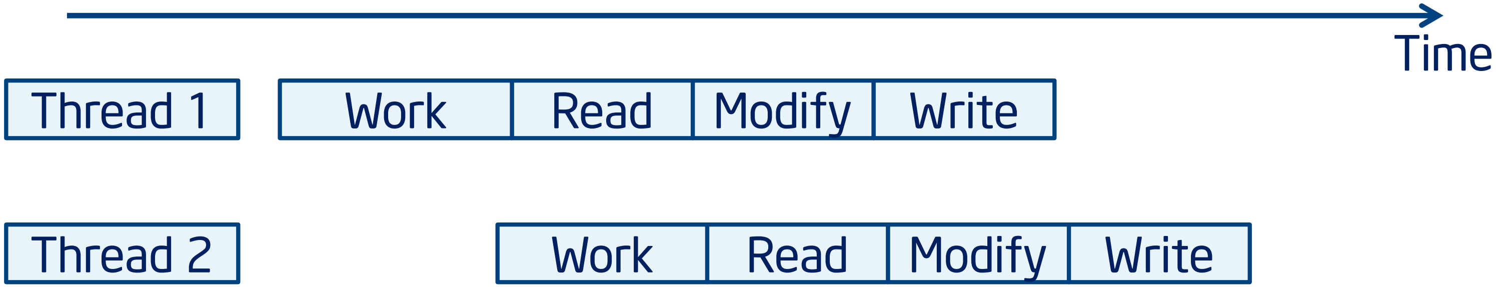


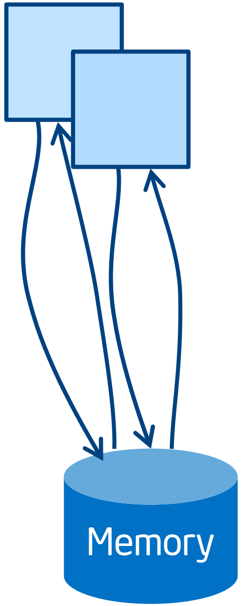
UAV

Back buffer

Volume-aware blending

- DirectX does not impose any ordering on the execution of pixel shader
    - Ordering happens later at the output merger stage
- If two threads read and modify the same memory, result is unpredictable

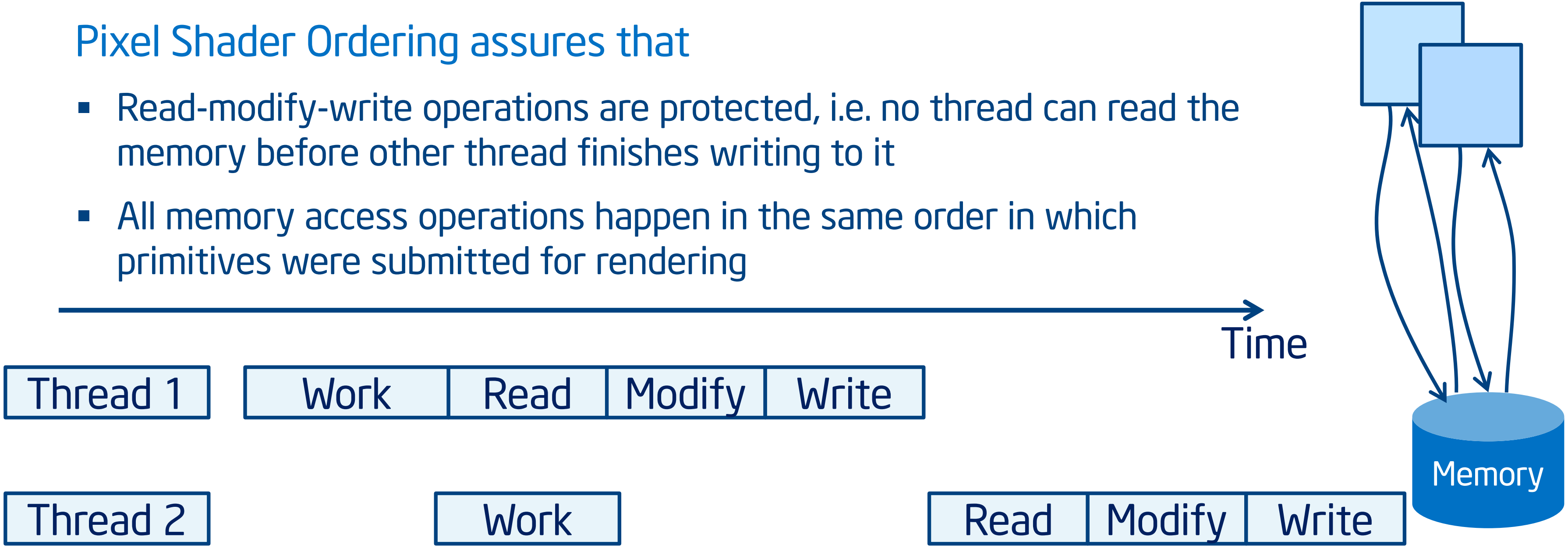




Volume-aware blending

Pixel Shader Ordering assures that

- Read-modify-write operations are protected, i.e. no thread can read the memory before other thread finishes writing to it
- All memory access operations happen in the same order in which primitives were submitted for rendering



Time

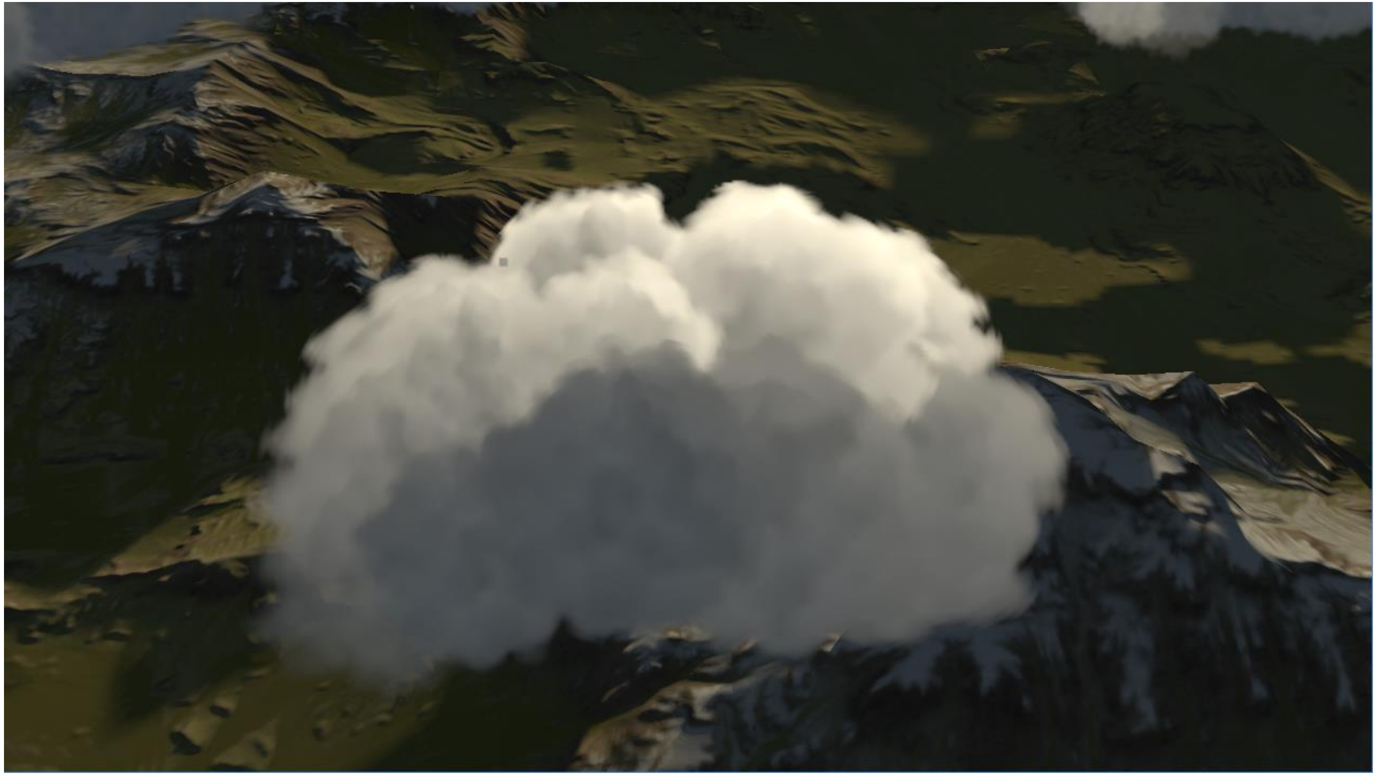
Thread 1 | Work | Read | Modify | Write

Thread 2 | Work | Read | Modify | Write

Memory

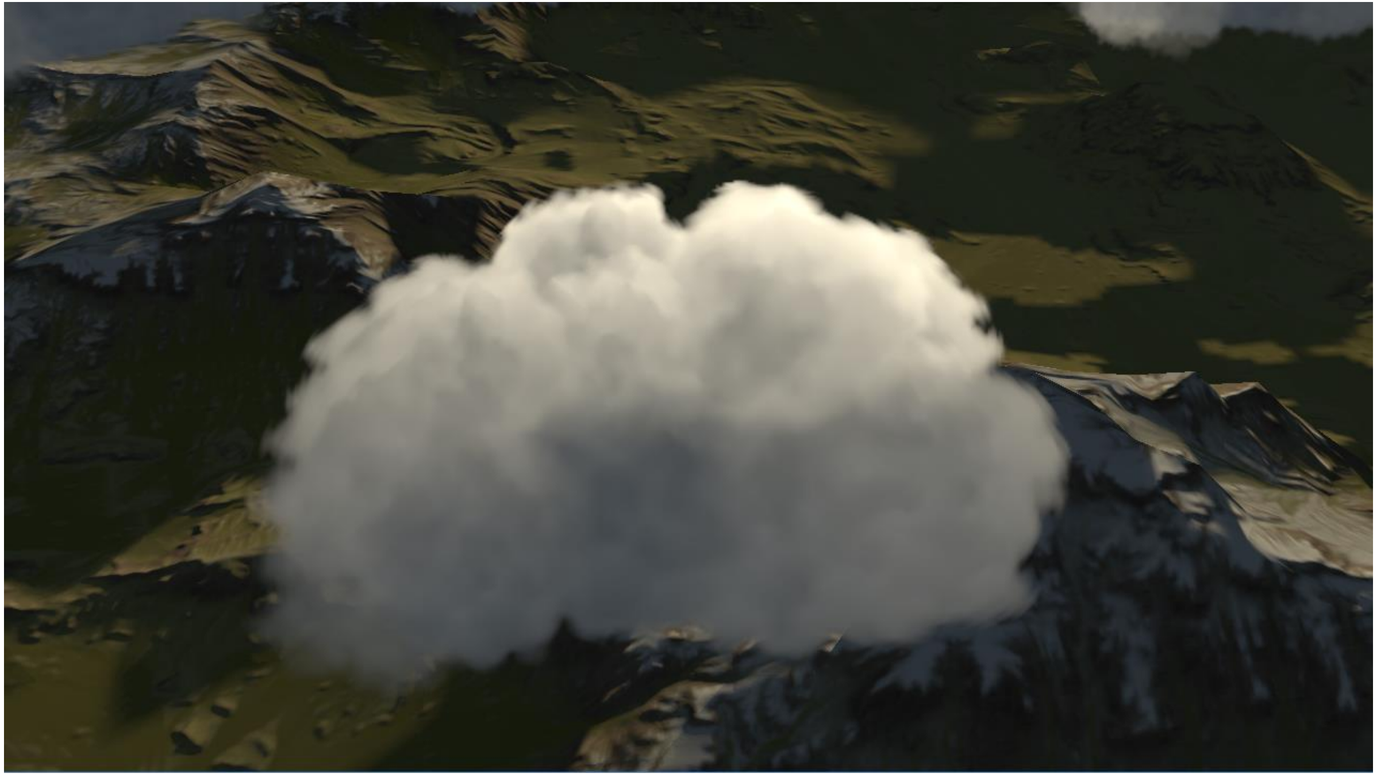
Volume-aware blending

No Pixel Sync – Conventional Alpha Blending



Volume-aware blending

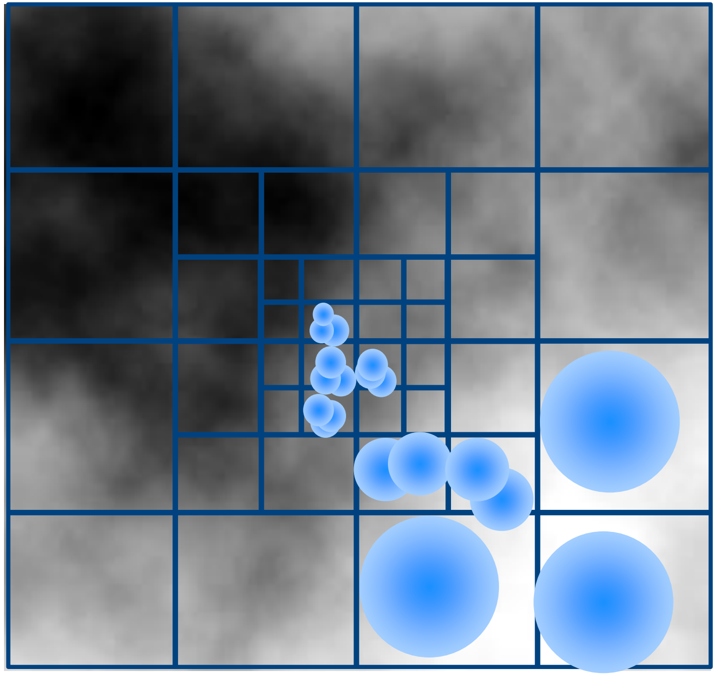
Pixel Sync – Volume-Aware Blending



Particle generation

Cell grid

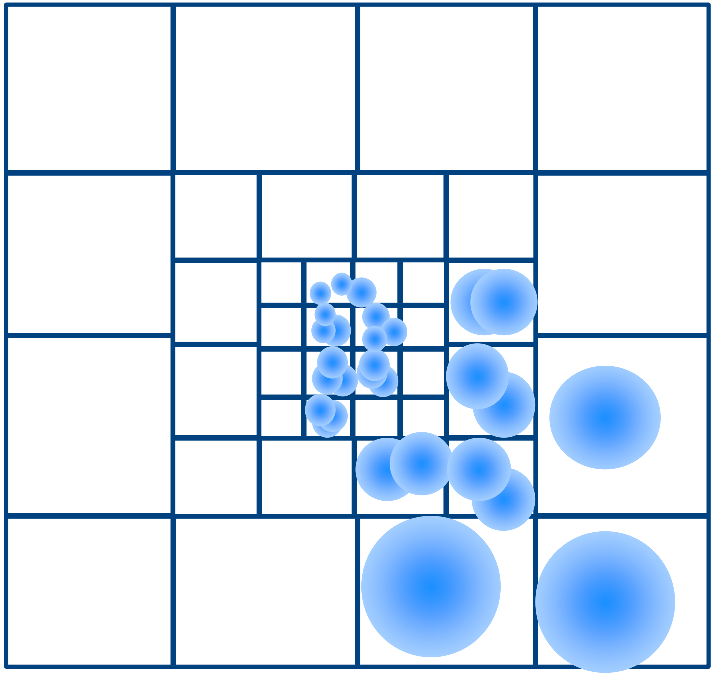
- Organized as a number of concentric rings centered around the camera
- Particles in each next ring have twice the size of the inner ring
- Each cell contains several layers of particles
- Density and size of particles in each cell are determined by the noise texture



Particle generation

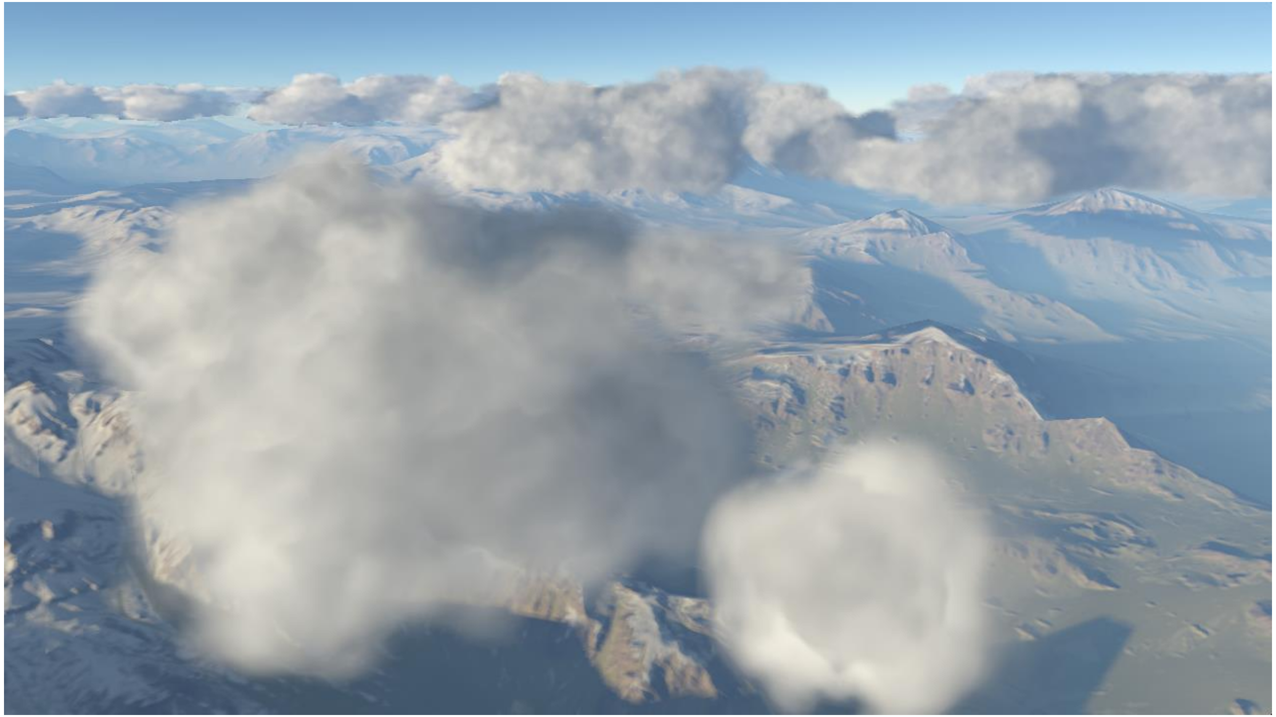
Animation:

Clouds are animated by changing particle size and transparency



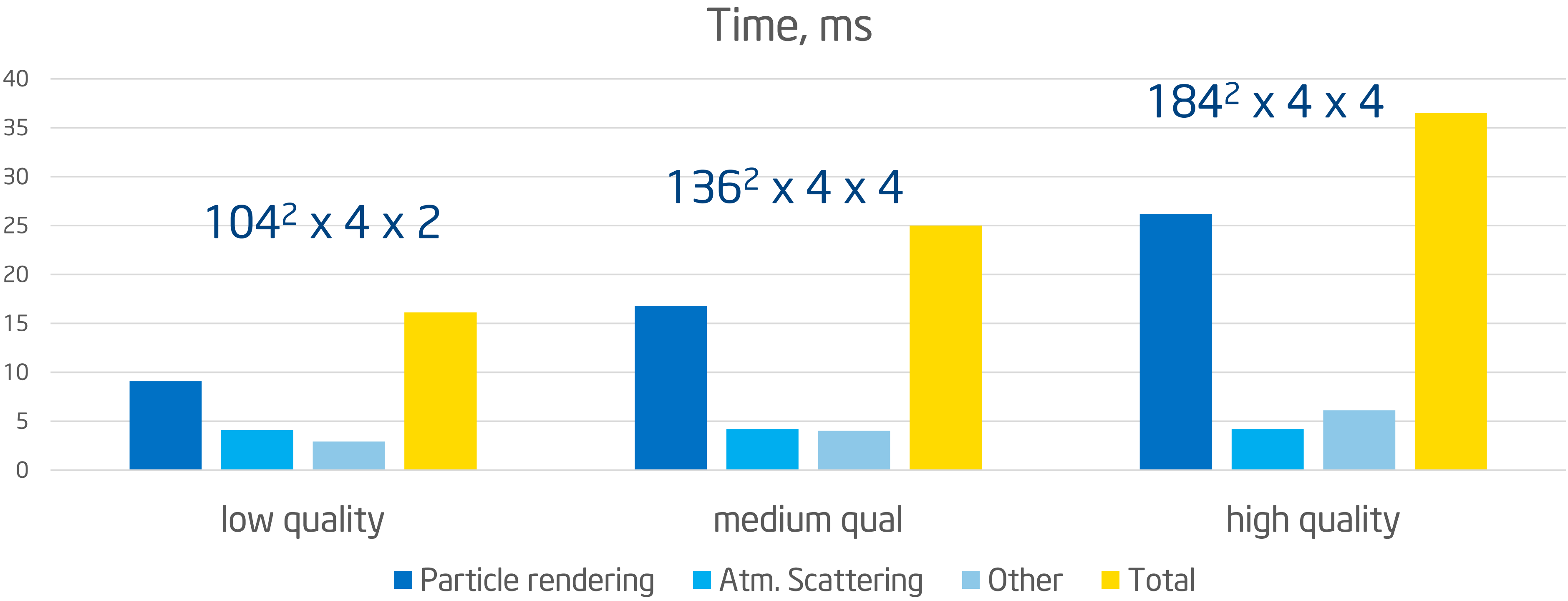
Results

Demo available at https://software.intel.com/en-us/blogs/2014/03/31/cloud-rendering-sample



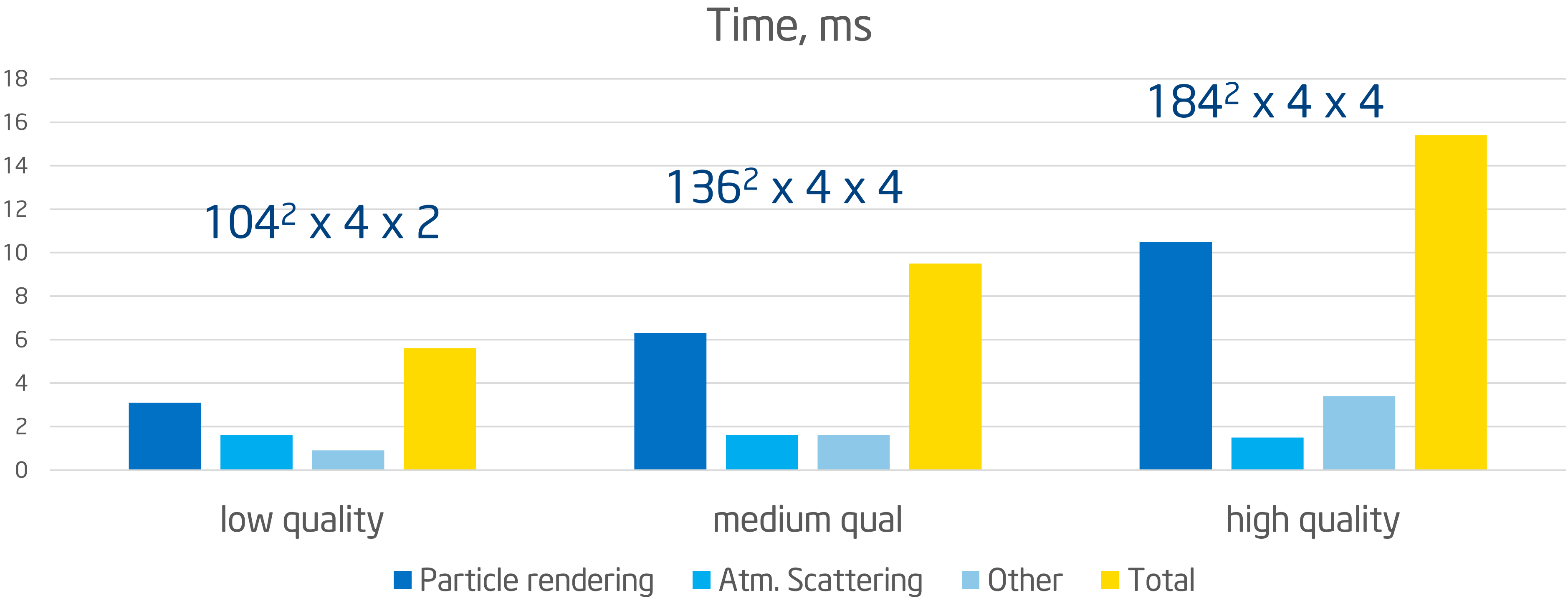
Performance

Intel Iris Pro 5200 (47 W), 1280x720



Performance

Nvidia GeForce GTX 680 (195 W), 1920x1080



Questions?

Thank You