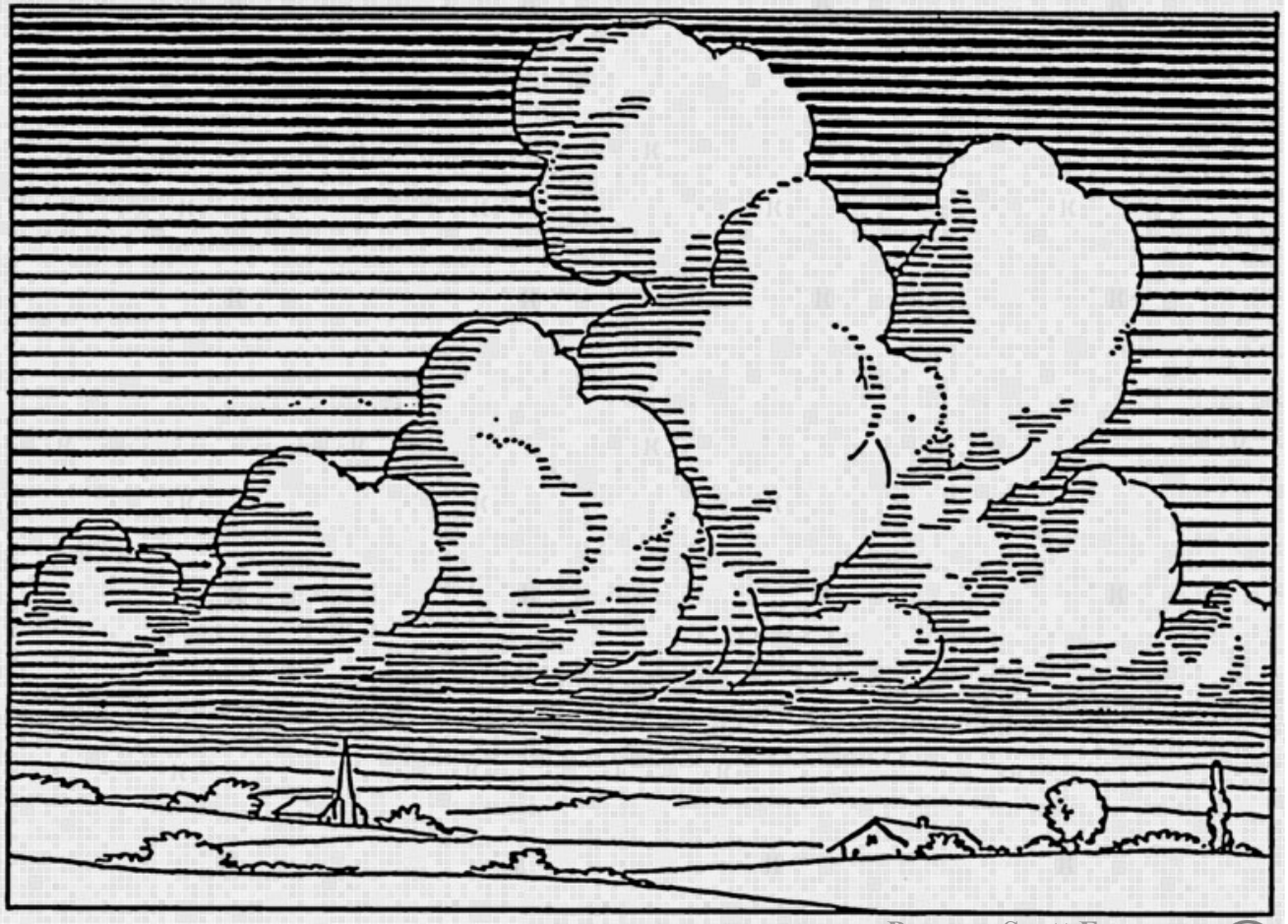
UNED

Escuela Internacional de Doctorado

EIDUNED



Pearson Scott Foresman

High-Performance Algorithms for Real-Time GPGPU Volumetric Cloud Rendering from an Enhanced Physical-Math Abstraction Approach

Carlos Jiménez de Parga Bernal - Quirós

Director: José A. Cerrada Somolinos

Co-director: Sebastián R. Gómez Palomo

INDEX

PART I

Problem statement

Target image to develop

Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality

Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality
- Realistic simulation requires powerful hardware equipment and many hours of rendering ⟶ Non real-time

Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality
- Realistic simulation requires powerful hardware equipment and many hours of rendering → Non real-time
- Today's real-time solutions requires massive multicore GPUs → expensive graphics cards

Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality
- Realistic simulation requires powerful hardware equipment and many hours of rendering ⟶ Non real-time
- Today's real-time solutions requires massive multicore GPUs ⟶ expensive graphics cards
- Currently there are limited solutions:

Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality
- Realistic simulation requires powerful hardware equipment and many hours of rendering ⟶ Non real-time
- Today's real-time solutions requires massive multicore GPUs ⟶ expensive graphics cards
- Currently there are limited solutions:
  - Texturized skydomes

Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality
- Realistic simulation requires powerful hardware equipment and many hours of rendering ⟶ Non real-time
- Today's real-time solutions requires massive multicore GPUs ⟶ expensive graphics cards
- Currently there are limited solutions:
  - Texturized skydomes
  - Particle systems

Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality
- Realistic simulation requires powerful hardware equipment and many hours of rendering ⟶ Non real-time
- Today's real-time solutions requires massive multicore GPUs ⟶ expensive graphics cards
- Currently there are limited solutions:
  - Texturized skydomes
  - Particle systems
  - Photo-realistic implementations

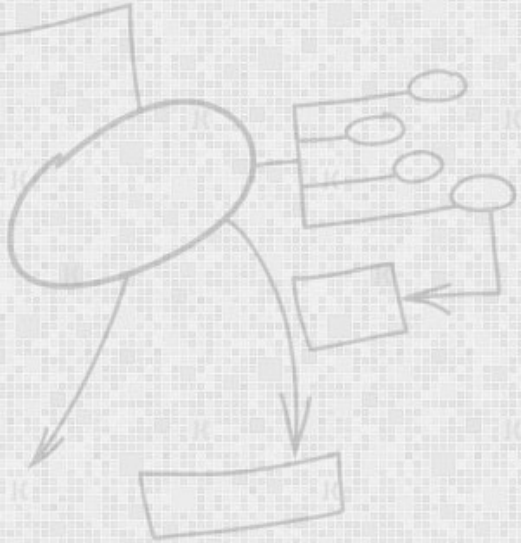
Problem statement

Target image to develop

- Very useful for game development, flight simulators and virtual reality
- Realistic simulation requires powerful hardware equipment and many hours of rendering ⟶ Non real-time
- Today's real-time solutions requires massive multicore GPUs ⟶ expensive graphics cards
- Currently there are limited solutions:
  - Texturized skydomes
  - Particle systems
  - Photo-realistic implementations
- An efficient volumetric rendering solution is required

Research objectives

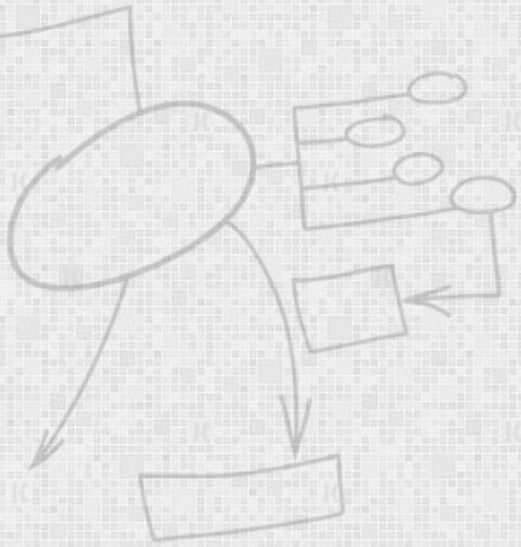
- High-performance
- Low detail



Research objectives

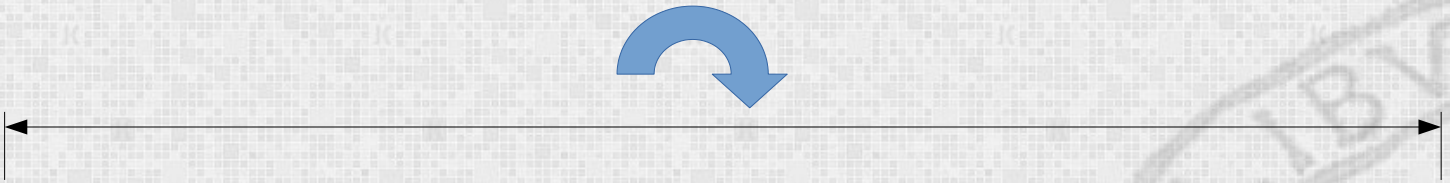
- High-performance
- Low detail

- Low-performance
- Hyper-realism



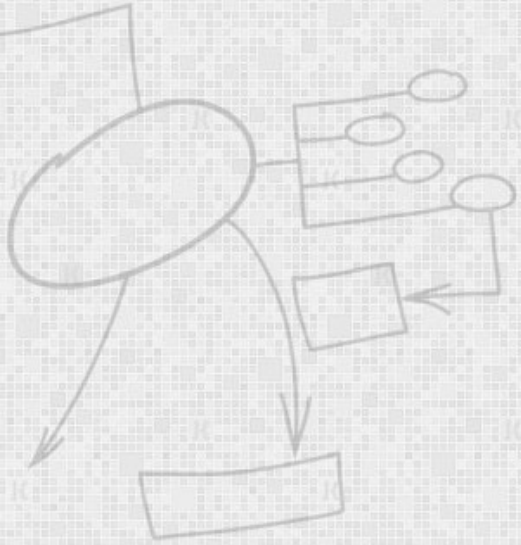
Research objectives

Thesis is here

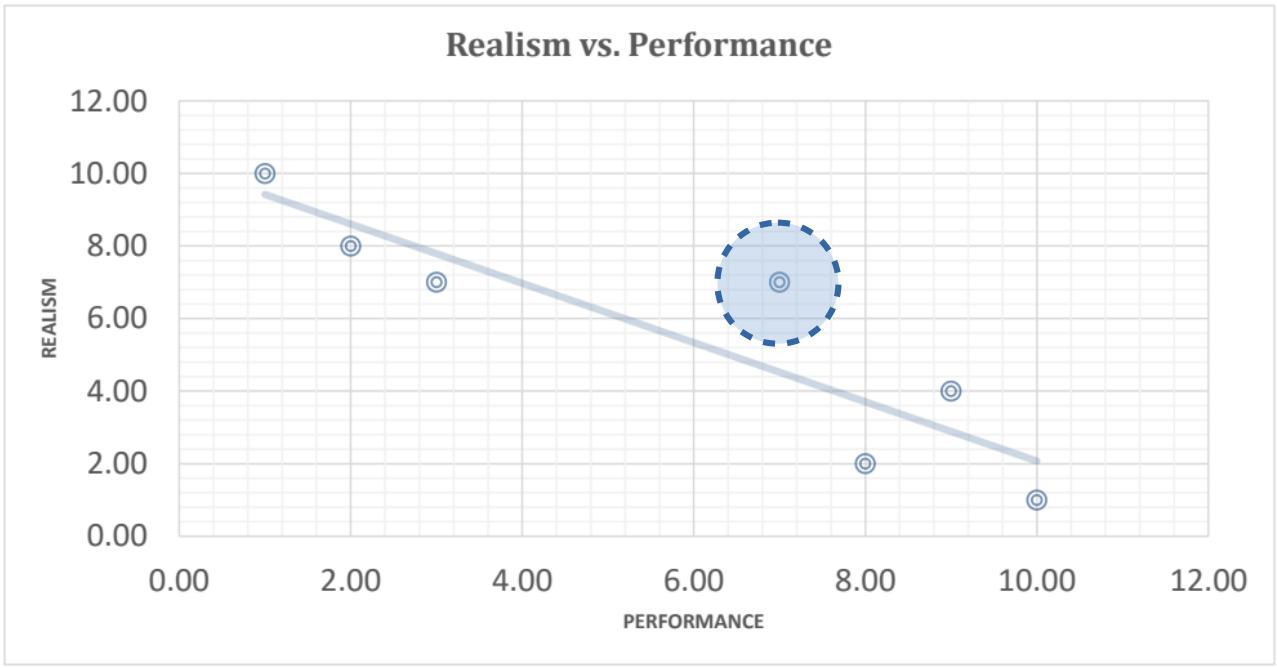


- High-performance
- Low detail

- Low-performance
- Hyper-realism

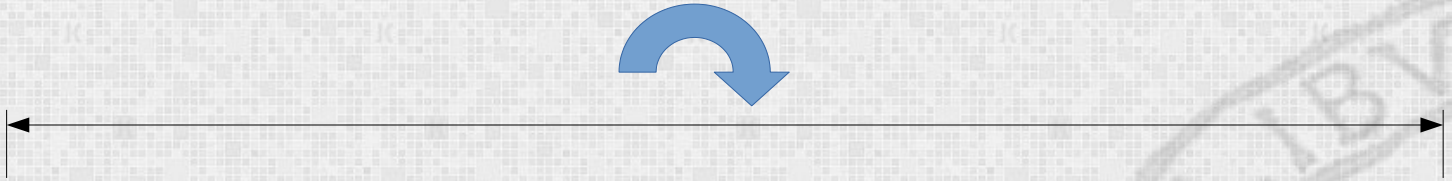


Research objectives



Research objectives

Thesis is here



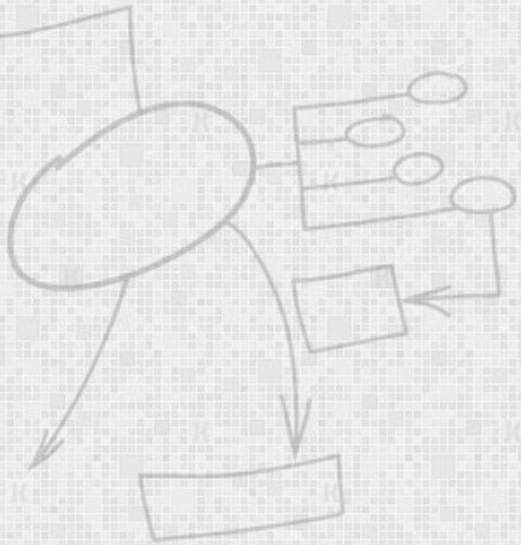
- High-performance
- Low detail

- Low-performance
- Hyper-realism

Hypothesis: «Is it possible to generate new algorithmic models for cloud rendering with an optimum balance between realism and performance to be applied in the entry-level graphics industry?».

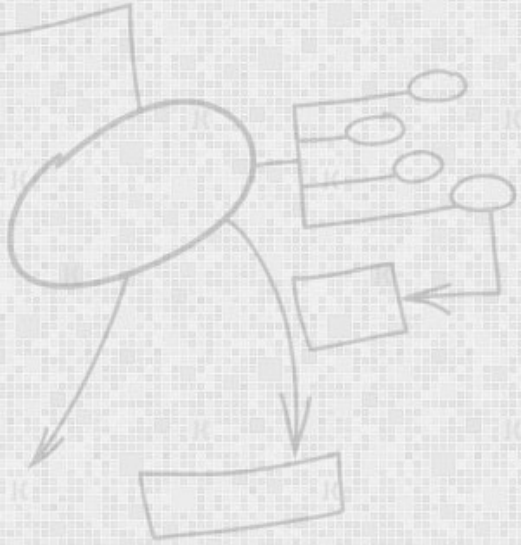
Overview of research contributions

- New method to generate the geometry of the cloud based on a minimal primitive known as pseudosphere that is replicated following different algorithms depending on the desired shape



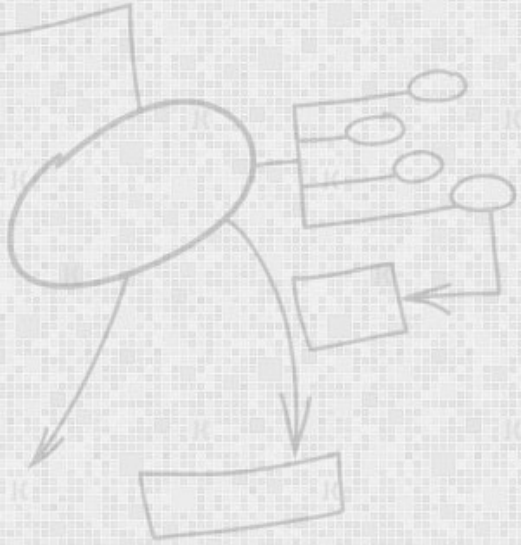
Overview of research contributions

- New method to generate the geometry of the cloud based on a minimal primitive known as pseudosphere that is replicated following different algorithms depending on the desired shape
- Lightweight pre-computing of shape and lighting in the CPU and GPU to minimize the load on processors with efficient volumetric rendering for light transmission and scattering



Overview of research contributions

- New method to generate the geometry of the cloud based on a minimal primitive known as pseudosphere that is replicated following different algorithms depending on the desired shape
- Lightweight pre-computing of shape and lighting in the CPU and GPU to minimize the load on processors with efficient volumetric rendering for light transmission and scattering
- Filtering algorithms to remove unnecessary computation caused by duplicate or void ray tracing outside camera frustum, excessive level of detail and superfluous volumetric rendering that has no contribution to realism



Overview of research contributions

- New method to generate the geometry of the cloud based on a minimal primitive known as pseudosphere that is replicated following different algorithms depending on the desired shape
- Lightweight pre-computing of shape and lighting in the CPU and GPU to minimize the load on processors with efficient volumetric rendering for light transmission and scattering
- Filtering algorithms to remove unnecessary computation caused by duplicate or void ray tracing outside camera frustum, excessive level of detail and superfluous volumetric rendering that has no contribution to realism
- Three innovative methods to generate the geometry of the cloud: Gaussian generator for cumulus, L-system generator for formal description of artistic cloud shapes and novel equations to produce clouds resembling known shapes described by 3D meshes

Overview of research contributions

- New method to generate the geometry of the cloud based on a minimal primitive known as pseudosphere that is replicated following different algorithms depending on the desired shape
- Lightweight pre-computing of shape and lighting in the CPU and GPU to minimize the load on processors with efficient volumetric rendering for light transmission and scattering
- Filtering algorithms to remove unnecessary computation caused by duplicate or void ray tracing outside camera frustum, excessive level of detail and superfluous volumetric rendering that has no contribution to realism
- Three innovative methods to generate the geometry of the cloud: Gaussian generator for cumulus, L-system generator for formal description of artistic cloud shapes and novel equations to produce clouds resembling known shapes described by 3D meshes
- A new approach of cumuliform cloud dynamics based on the GPU and CPU load distribution to solve the Navier-Stokes fluid dynamics equations with high performance and reliable results

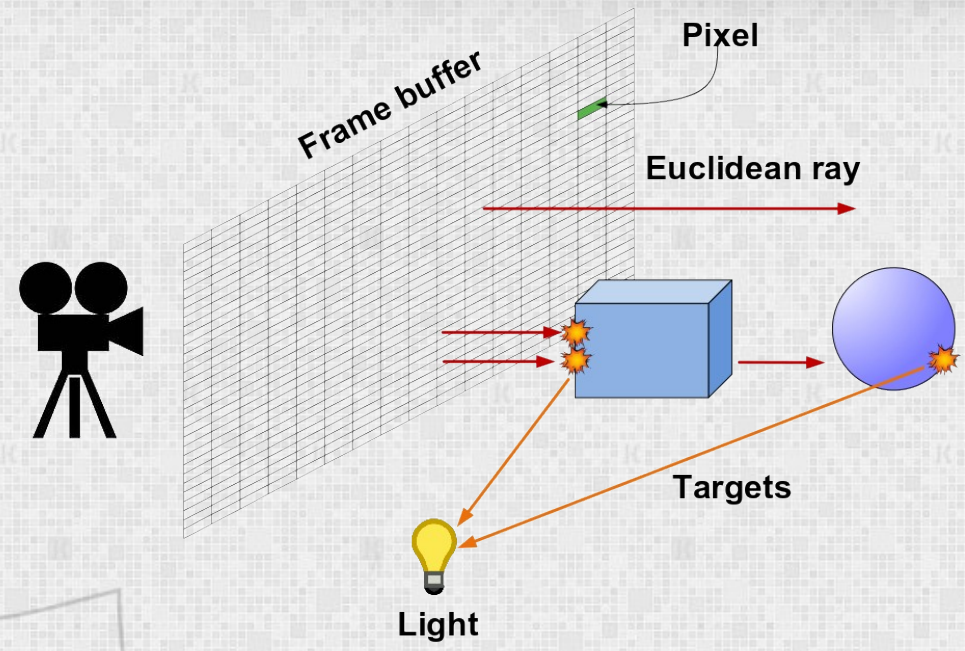
Overview of research contributions

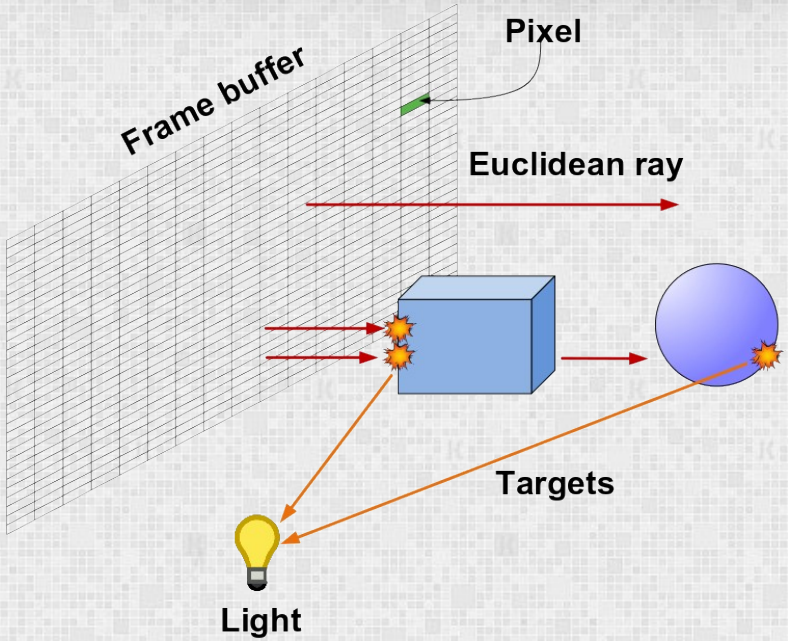
- New method to generate the geometry of the cloud based on a minimal primitive known as pseudosphere that is replicated following different algorithms depending on the desired shape
- Lightweight pre-computing of shape and lighting in the CPU and GPU to minimize the load on processors with efficient volumetric rendering for light transmission and scattering
- Filtering algorithms to remove unnecessary computation caused by duplicate or void ray tracing outside camera frustum, excessive level of detail and superfluous volumetric rendering that has no contribution to realism
- Three innovative methods to generate the geometry of the cloud: Gaussian generator for cumulus, L-system generator for formal description of artistic cloud shapes and novel equations to produce clouds resembling known shapes described by 3D meshes
- A new approach of cumuliform cloud dynamics based on the GPU and CPU load distribution to solve the Navier-Stokes fluid dynamics equations with high performance and reliable results
- A morphing algorithm

Raytracing

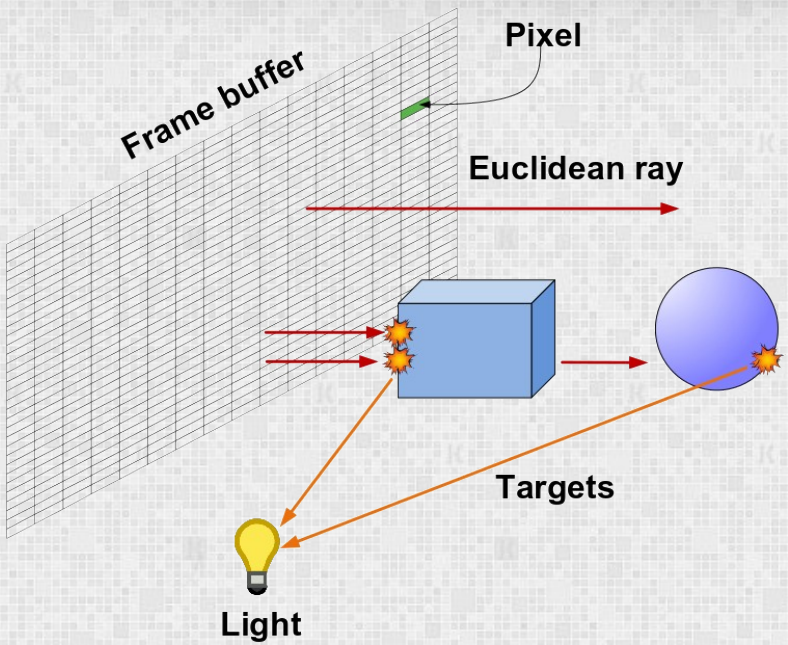
Albrecht Dürer (XVI c.)

Turner Whitted / Arthur Appel





$$R \equiv \begin{cases} x = x_0 + \lambda \vec{v}_x \\ y = y_0 + \lambda \vec{v}_y \\ z = z_0 + \lambda \vec{v}_z \end{cases} \qquad (1.1)$$



$$R \equiv \begin{cases} x = x_0 + \lambda \vec{v}_x \\ y = y_0 + \lambda \vec{v}_y \\ z = z_0 + \lambda \vec{v}_z \end{cases} \quad \textbf{(1.1)}$$

Let

$$(x-c_x)^2+(y-c_y)^2+(z-c_z)^2=R^2 \quad \textbf{(1.2)}$$

be the implicit equation of a sphere. We can rewrite this same equation in vector form as:

$$(p-c)\cdot(p-c)=R^2 \quad \textbf{(1.3)}$$

If we assume the vector form of Equation **1.1** as $p(t)=o+\lambda\vec{v}$ and substitute it into Equation **1.3**, we obtain:

$$(o+\lambda\vec{v}-c)\cdot(o+\lambda\vec{v}-c)=R^2 \quad \textbf{(1.4)}$$

Moving the terms around yields

$$(\vec{v}\cdot\vec{v})\lambda^2+2\vec{v}\cdot(o-c)\lambda+(o-c)\cdot(o-c)-R^2=0 \quad \textbf{(1.5)}$$

Equation **1.5** constitutes a classic quadratic equation in λ:

$$A\lambda^2+B\lambda+C=0; \quad \textbf{(1.6)}$$

It is known that this equation has two solutions:

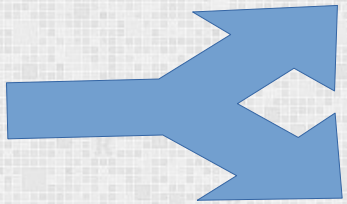
$$\lambda=\frac{-B\pm\sqrt{B^2-4AC}}{2A} \quad \textbf{(1.7)}$$

where the discriminant is:

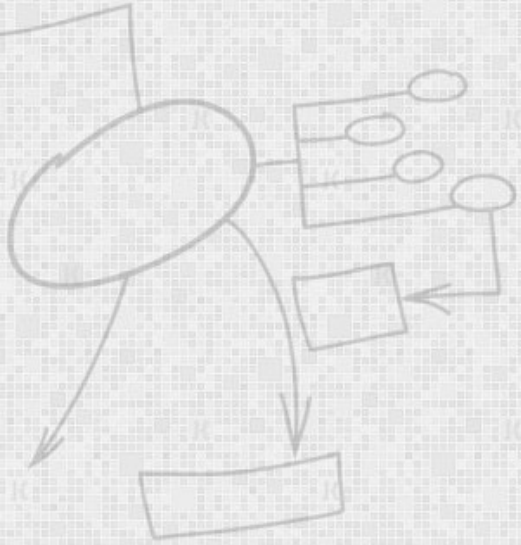
$$\Delta=(2\vec{v}\cdot(o-c))^2-4(\vec{v}\cdot\vec{v})((o-c)\cdot(o-c)-R^2) \quad \textbf{(1.8)}$$

Raymarching / Raycasting

Raytracing variants

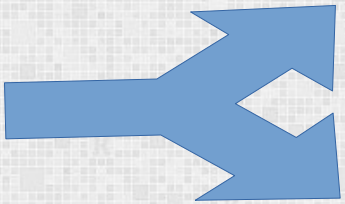


- Raymarching
- Raycasting

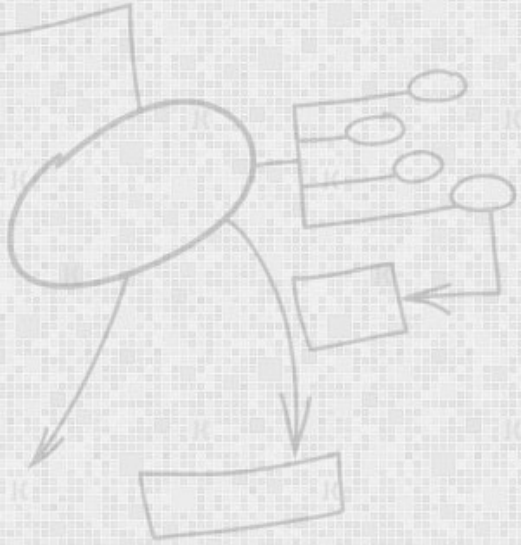


Raymarching / Raycasting

Raytracing variants

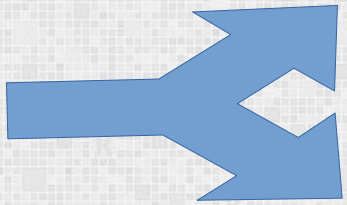


- Raymarching
  - Lightweight
  - Discrete sampling
  - Used in this thesis
- Raycasting

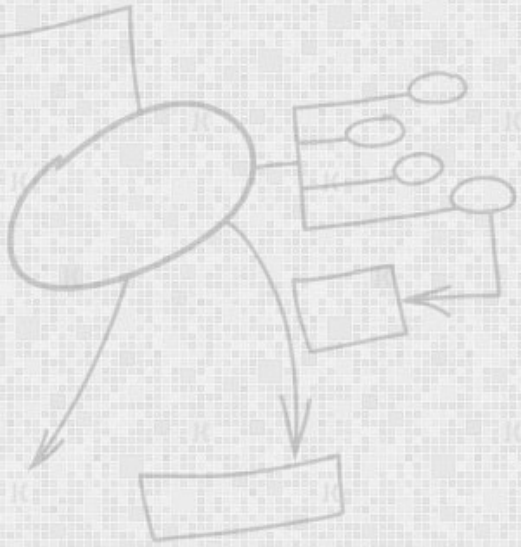


Raymarching / Raycasting

Raytracing variants

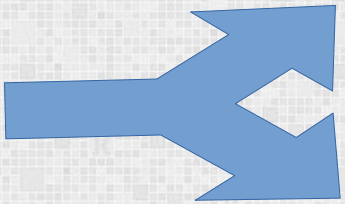


- Raymarching
  - Lightweight
  - Discrete sampling
  - Used in this thesis
- Raycasting
  - Analytical calculations
  - Use voxels
  - MRI and CT
  - Used in this thesis

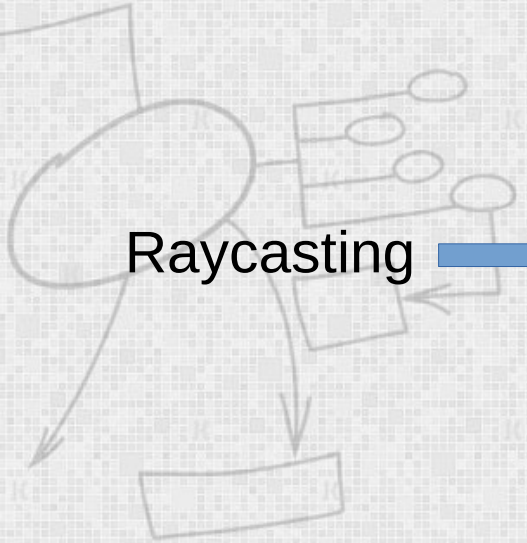


Raymarching / Raycasting

Raytracing variants



- Raymarching
  - Lightweight
  - Discrete sampling
  - Used in this thesis
- Raycasting
  - Analytical calculations
  - Use voxels
  - MRI and CT
  - Used in this thesis



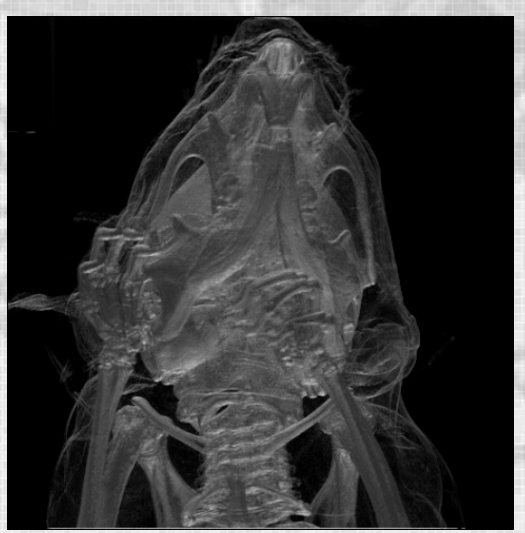
Raycasting → Volume rendering (CT)

Raymarching / Raycasting

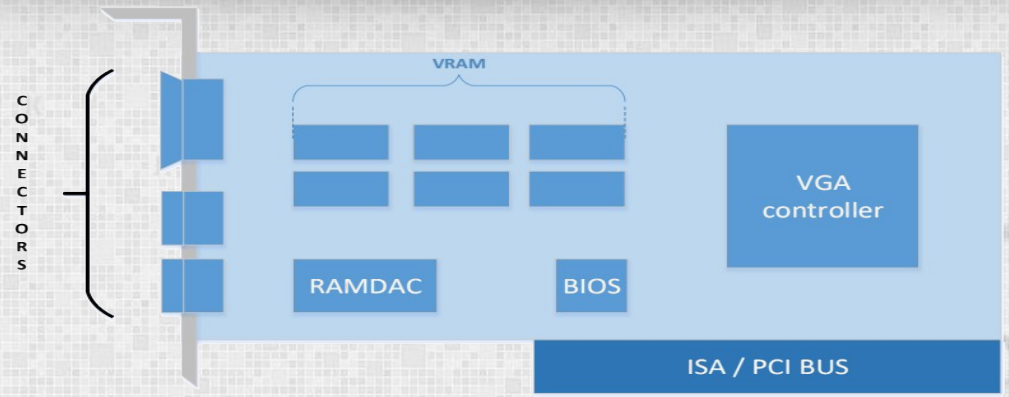
Raytracing variants

- Raymarching
  - Lightweight
  - Discrete sampling
  - Used in this thesis
- Raycasting
  - Analytical calculations
  - Use voxels
  - MRI and CT
  - Used in this thesis

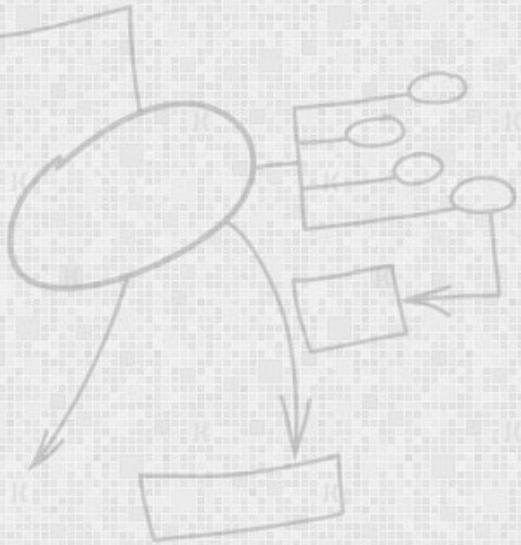
Raycasting → Volume rendering (CT) →



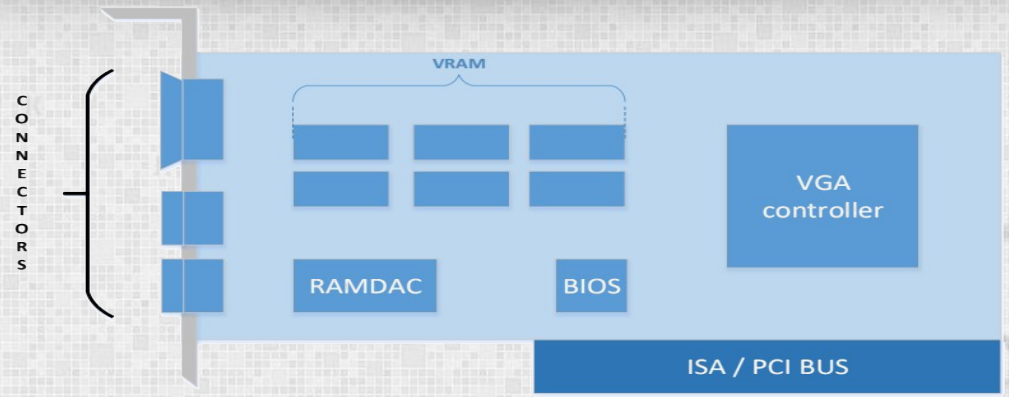
GPU rendering pipeline



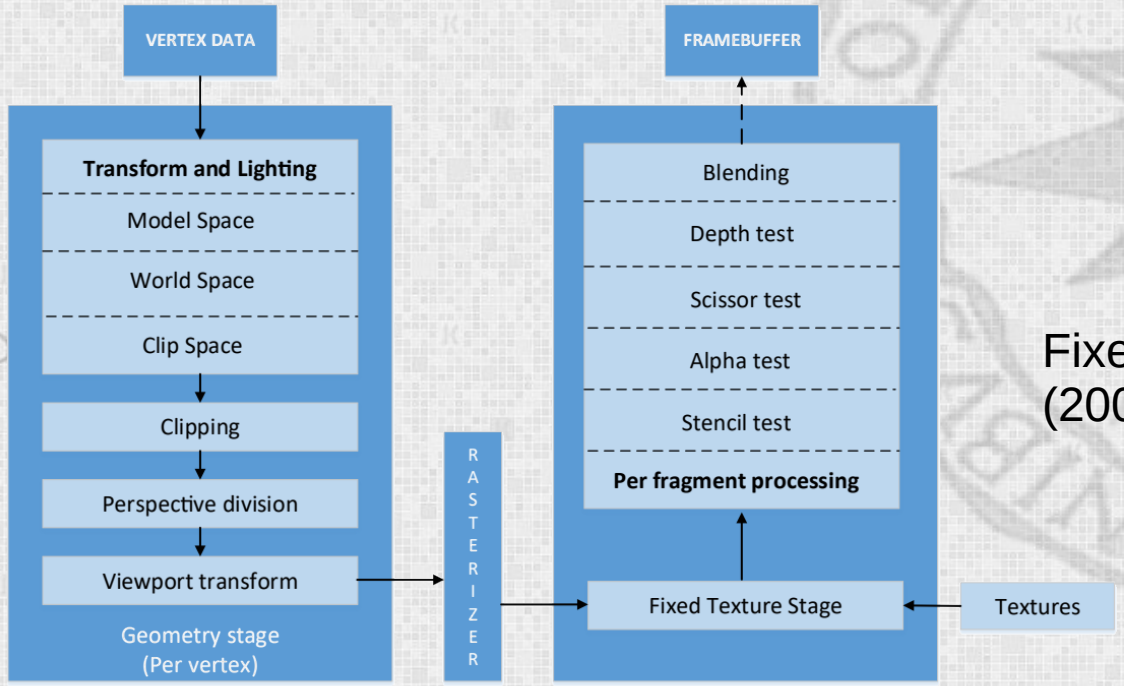
VGA classic card
(1989 - 2002)



GPU rendering pipeline

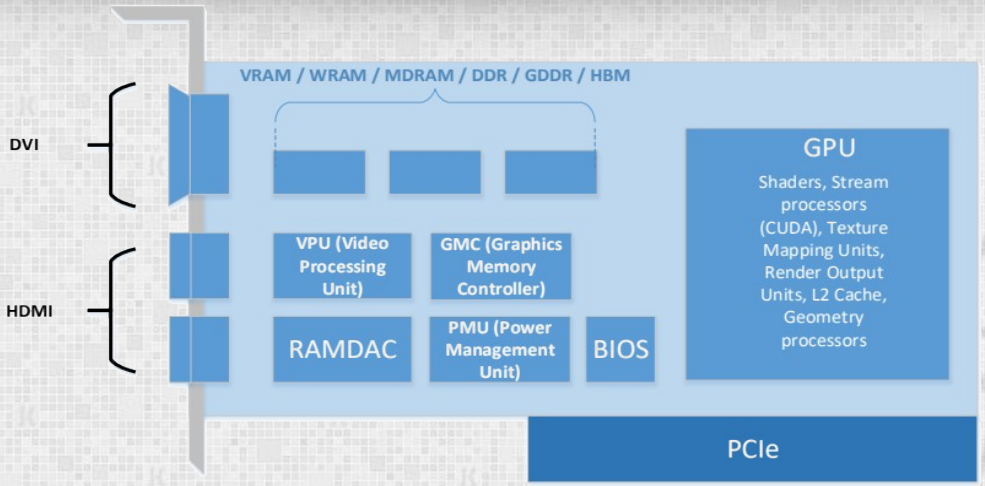


VGA classic card
(1989 - 2002)

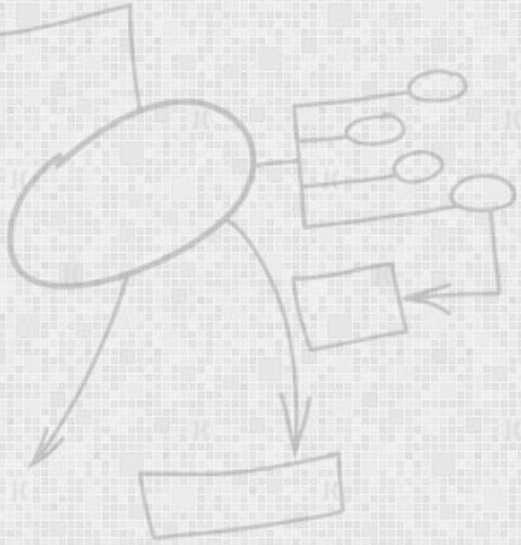


Fixed pipeline
(2002 and earlier)

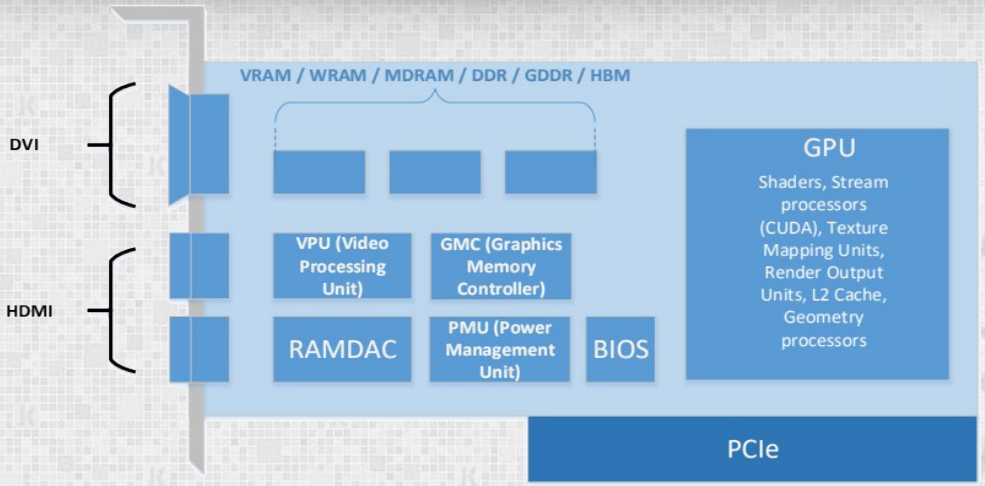
GPU rendering pipeline



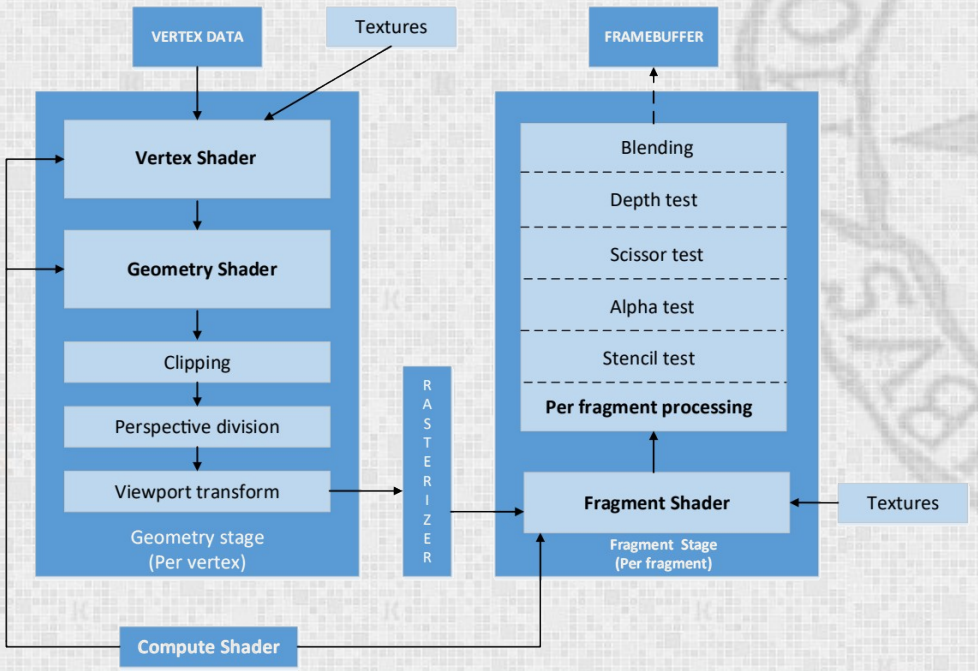
A GPU graphics card (2002 to present)



GPU rendering pipeline



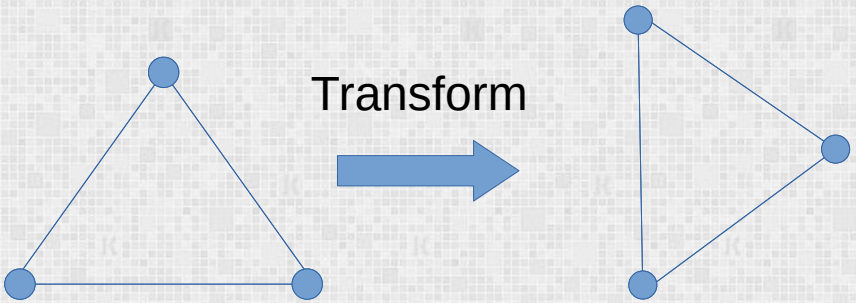
A GPU graphics card (2002 to present)

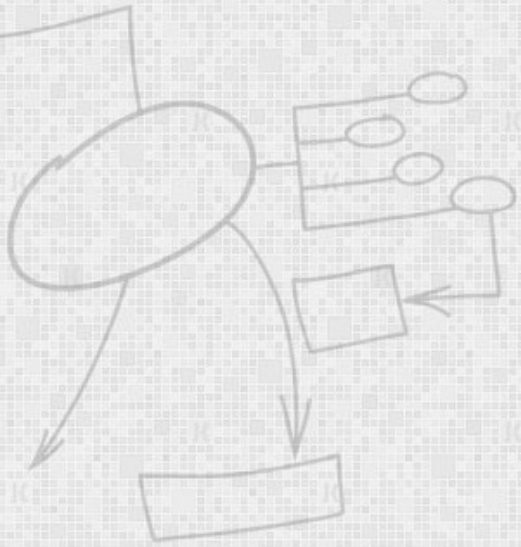


Programmable graphics pipeline (2004 to present)

Shader types

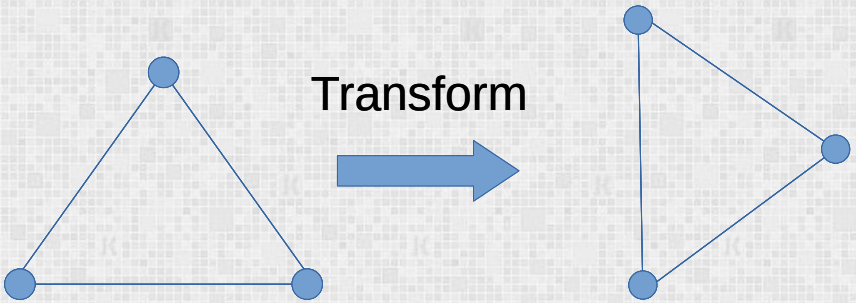
- **Vertex shaders:** Geometry transformation, MVP, calculate normals, etc.



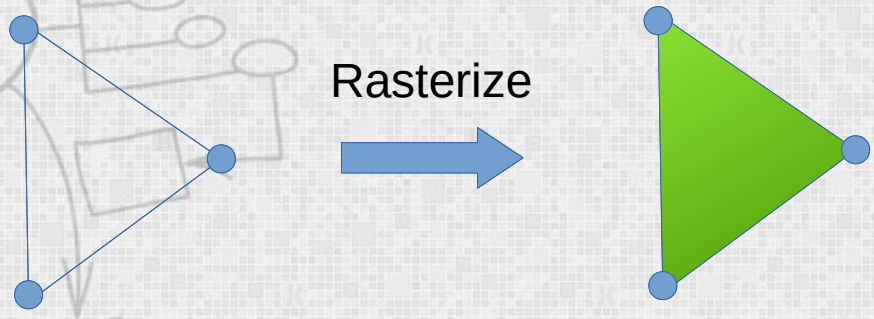


Shader types

- **Vertex shaders:** Geometry transformation, MVP, calculate normals, etc.

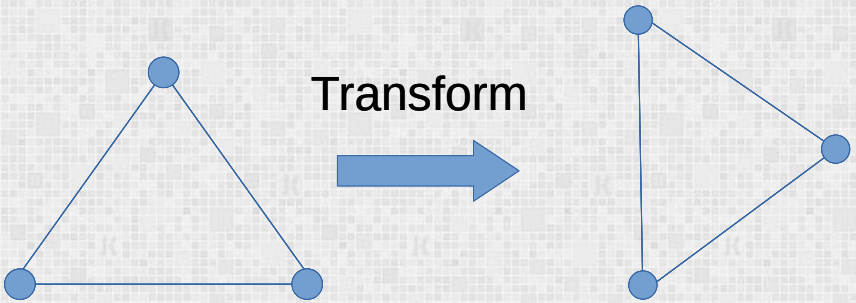


- **Fragment shaders:** Fills primitives according to RGBA values

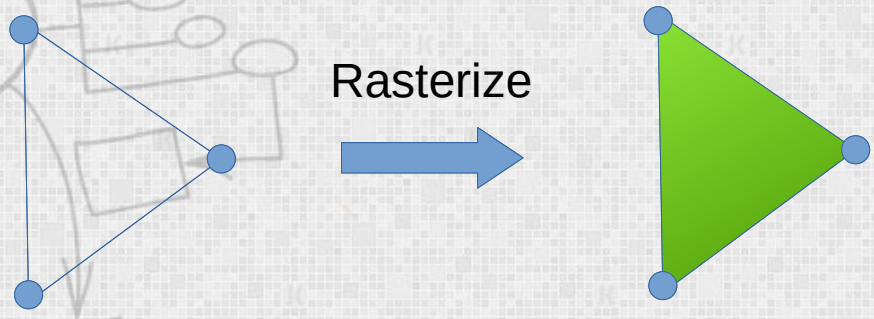


Shader types

- **Vertex shaders:** Geometry transformation, MVP, calculate normals, etc.



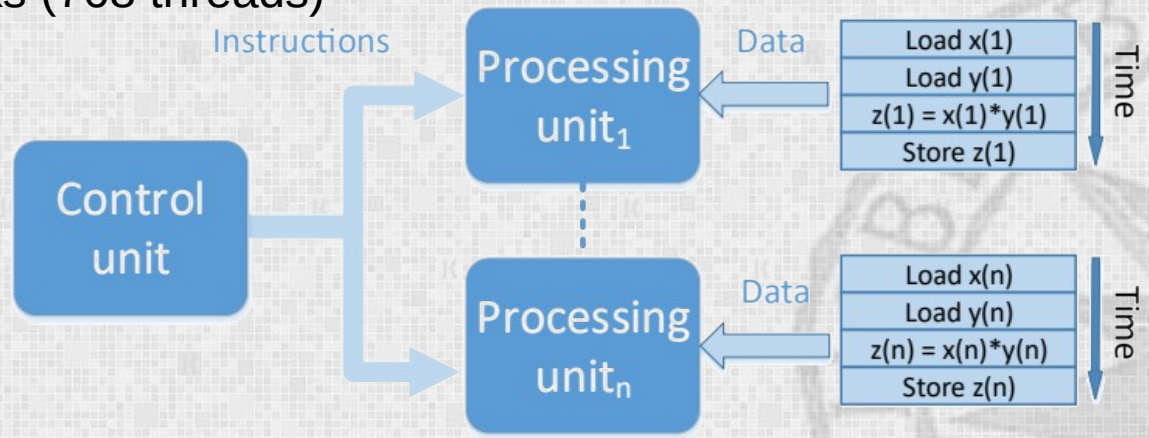
- **Fragment shaders:** Fills primitives according to RGBA values



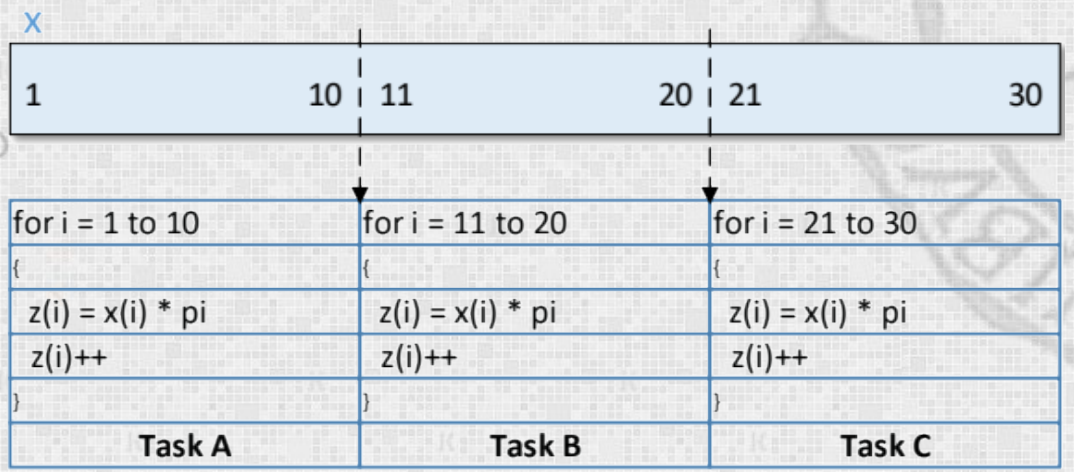
Go through the GPU pipeline

GPGPU programming

- GPGPU parallel programming capabilities thanks to nVidia CUDA
- The CUDA GPU contains N^3 *streaming multiprocessors*, where each multiprocessor is a set of 8 streaming processors (SPs). Each SP creates, plans and executes up to 24 *warps* in one or more blocks (768 threads)



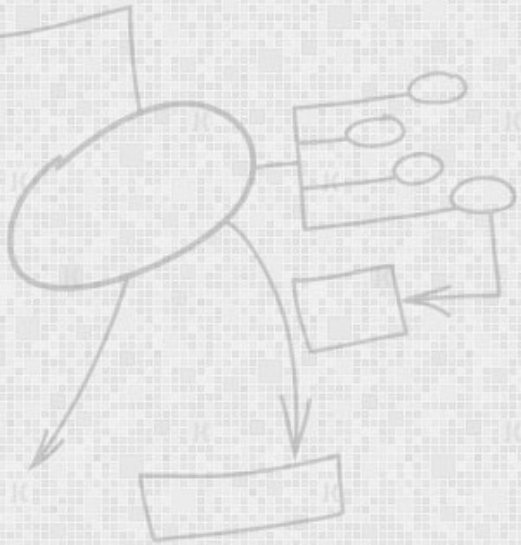
SIMD model



Data level parallel programming example

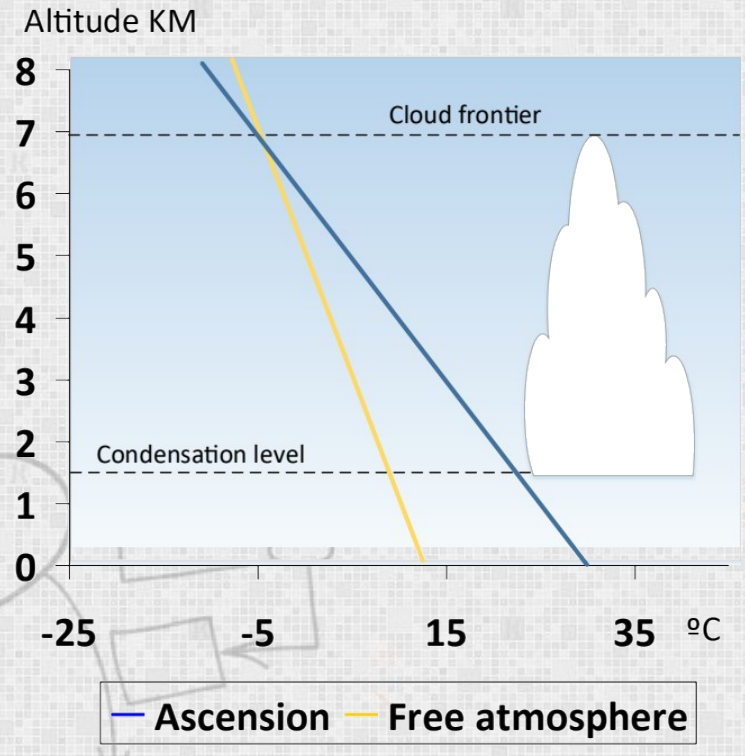
Cloud formation

According to Hans Häckel [Häc06], the air contains water but in vapour form. When air containing water vapour comes in contact with a sufficiently cold temperature, the small water droplets are formed in the natural process called *condensation*



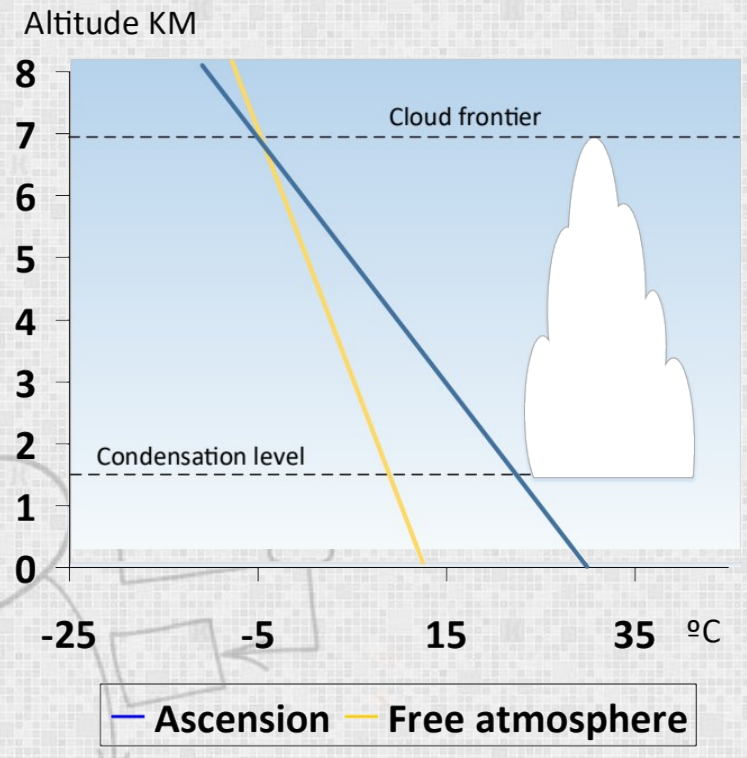
Cloud formation

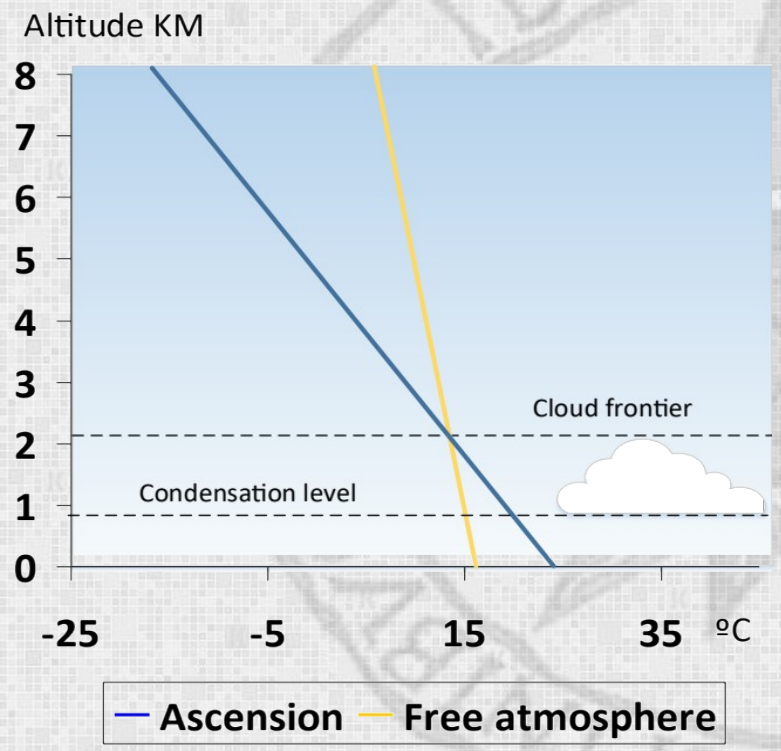
According to Hans Häckel [Häc06], the air contains water but in vapour form. When air containing water vapour comes in contact with a sufficiently cold temperature, the small water droplets are formed in the natural process called *condensation*



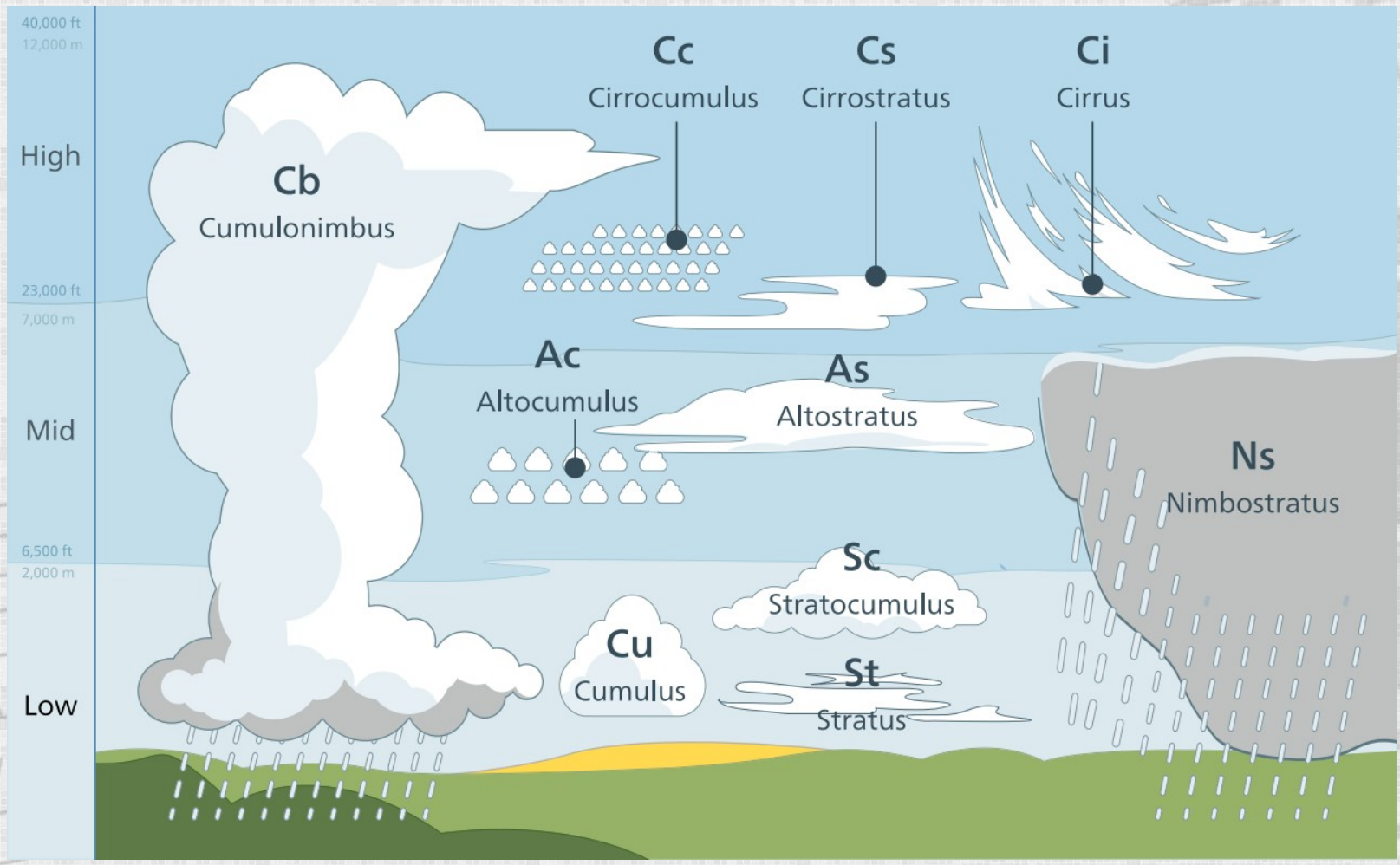
Cloud formation

According to Hans Häckel [Häc06], the air contains water but in vapour form. When air containing water vapour comes in contact with a sufficiently cold temperature, the small water droplets are formed in the natural process called *condensation*



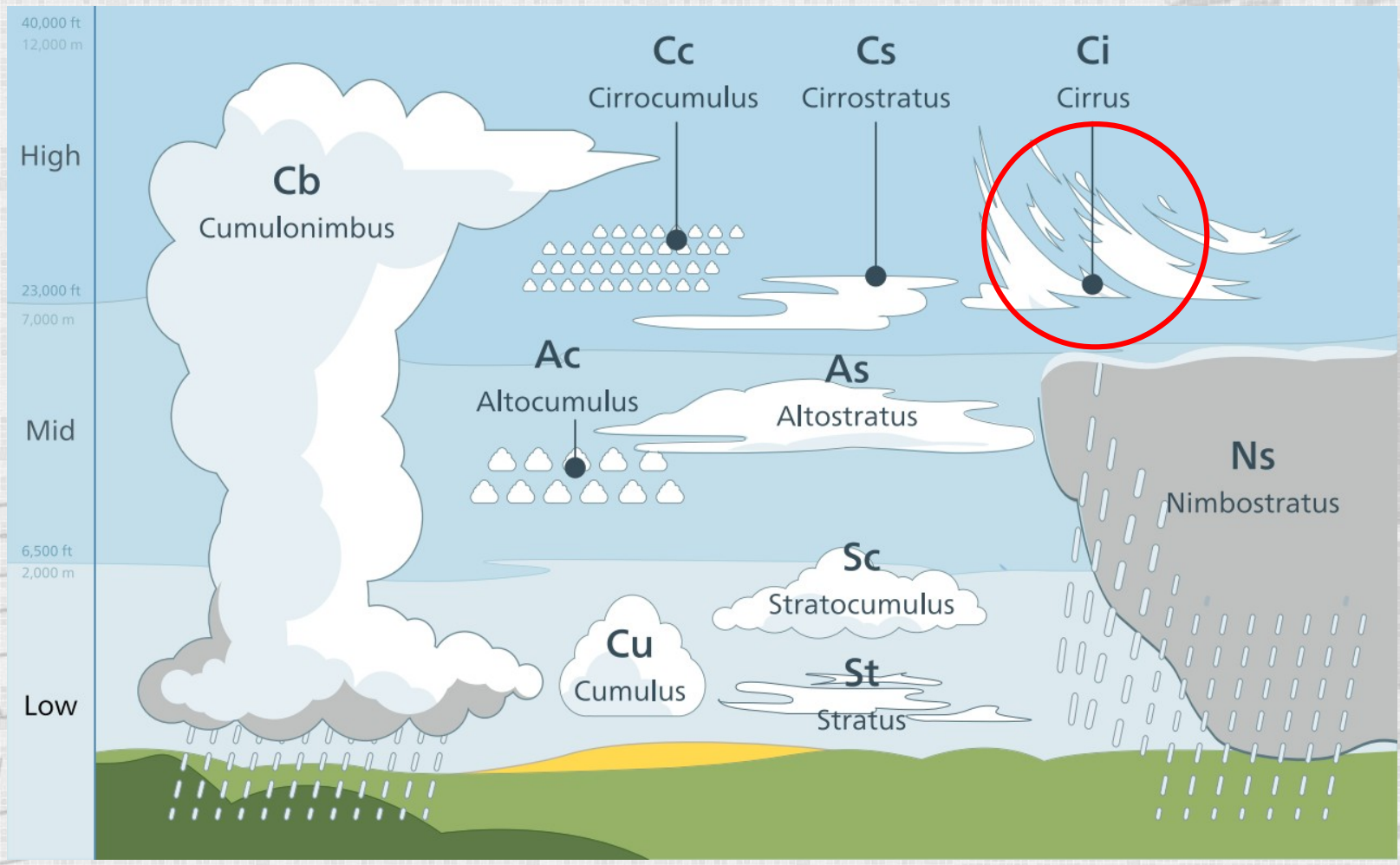


Cloud taxonomy



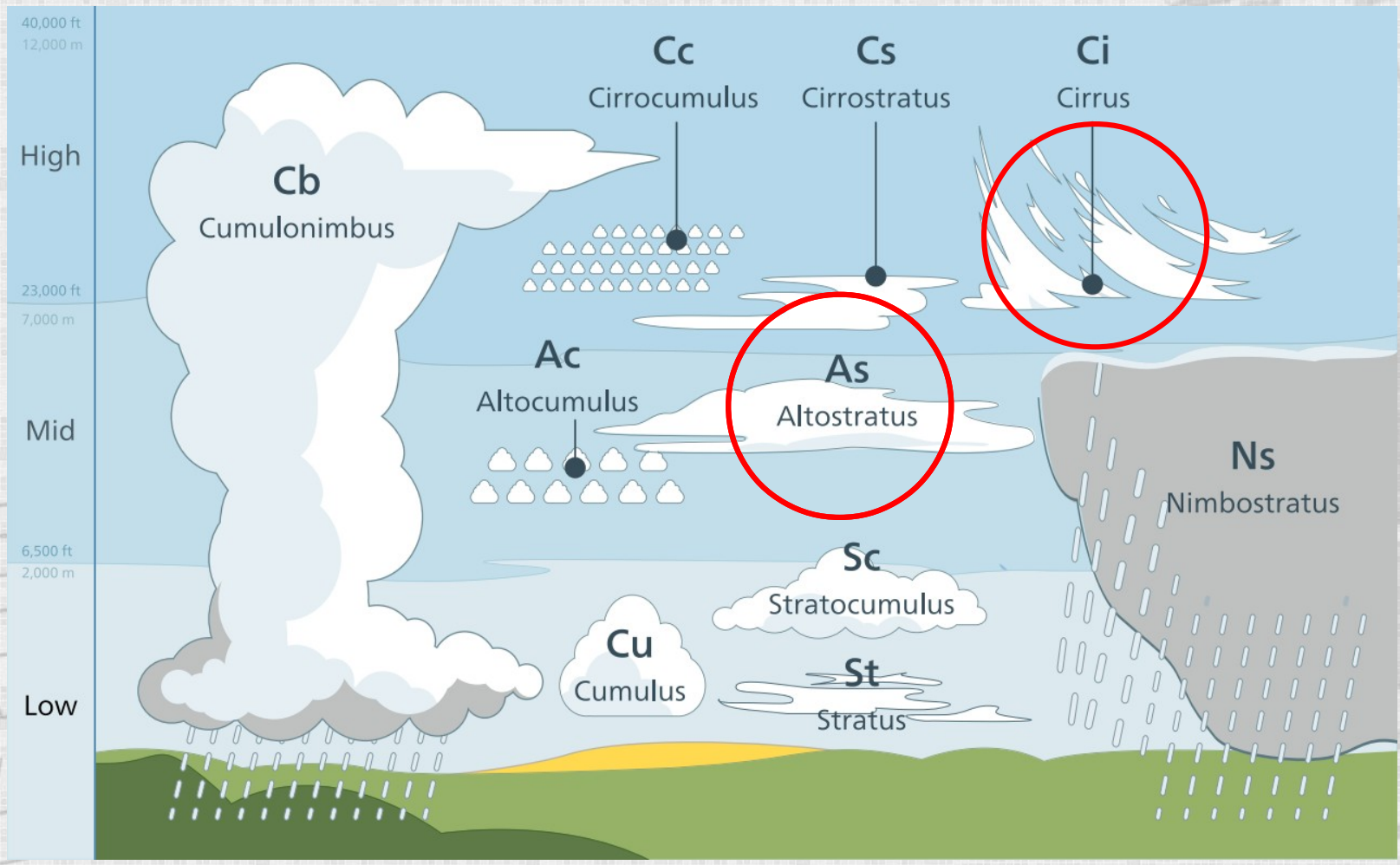
Cloud taxonomy

Clouds modelled in the current thesis:



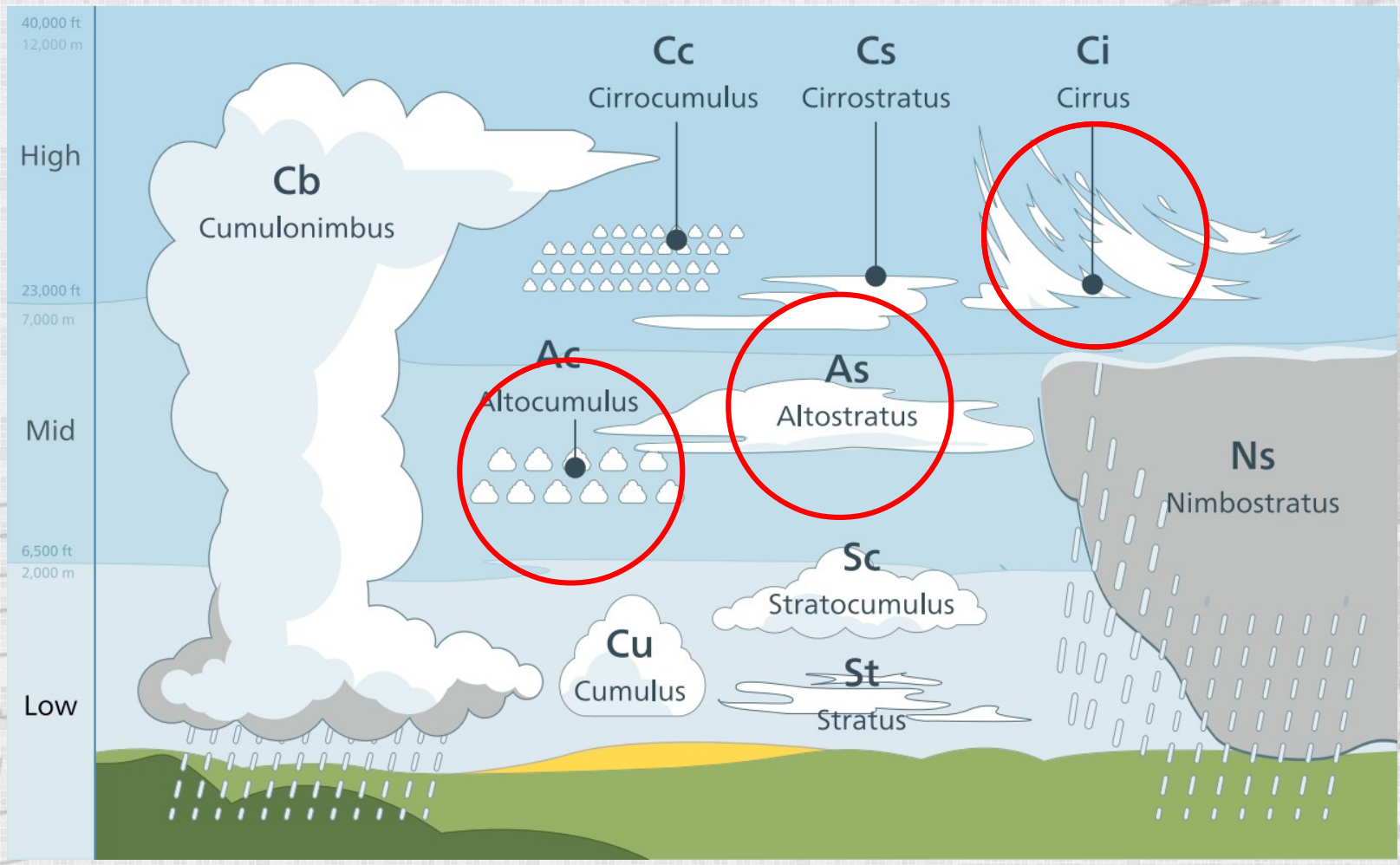
Cloud taxonomy

Clouds modelled in the current thesis:



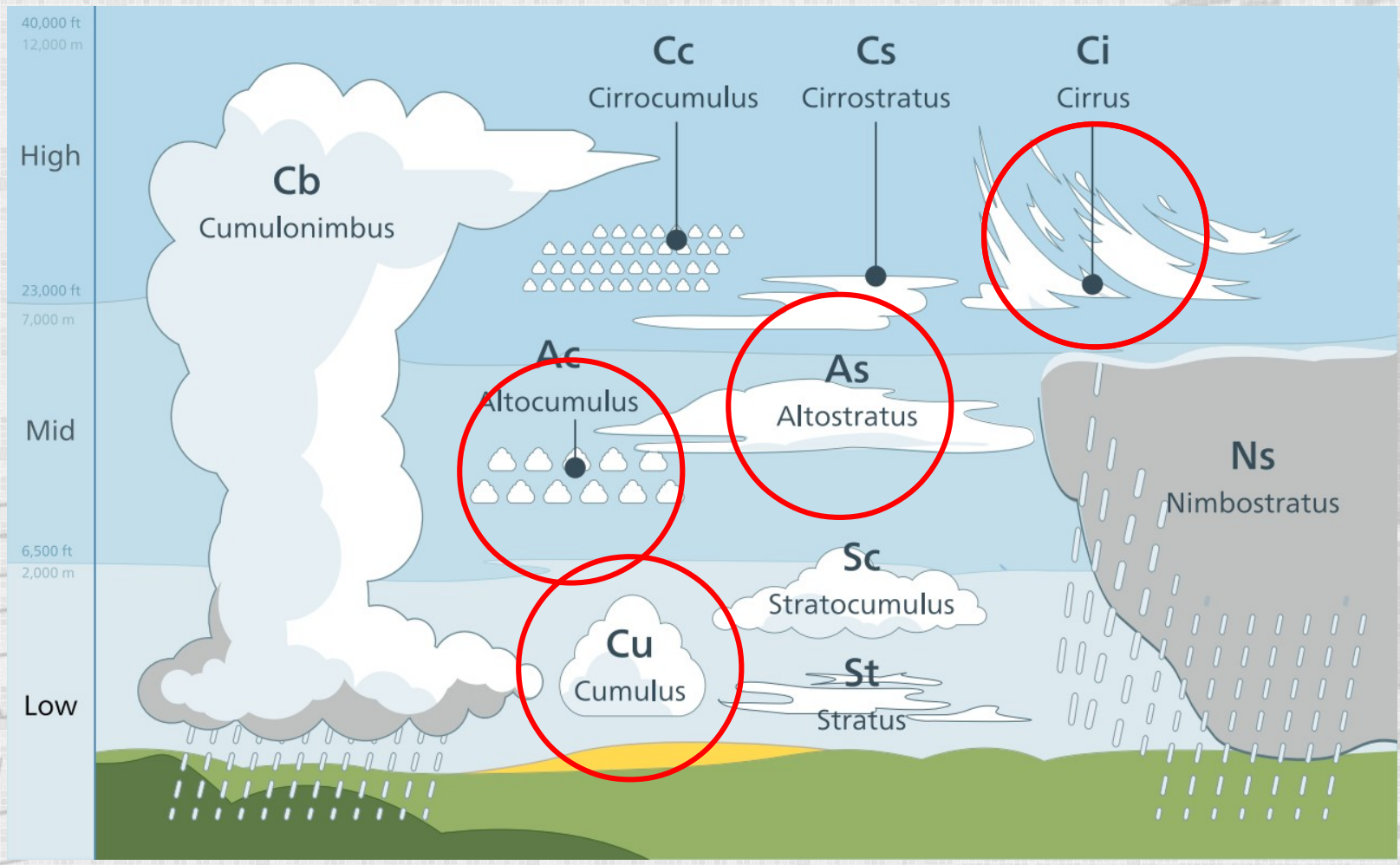
Cloud taxonomy

Clouds modelled in the current thesis:



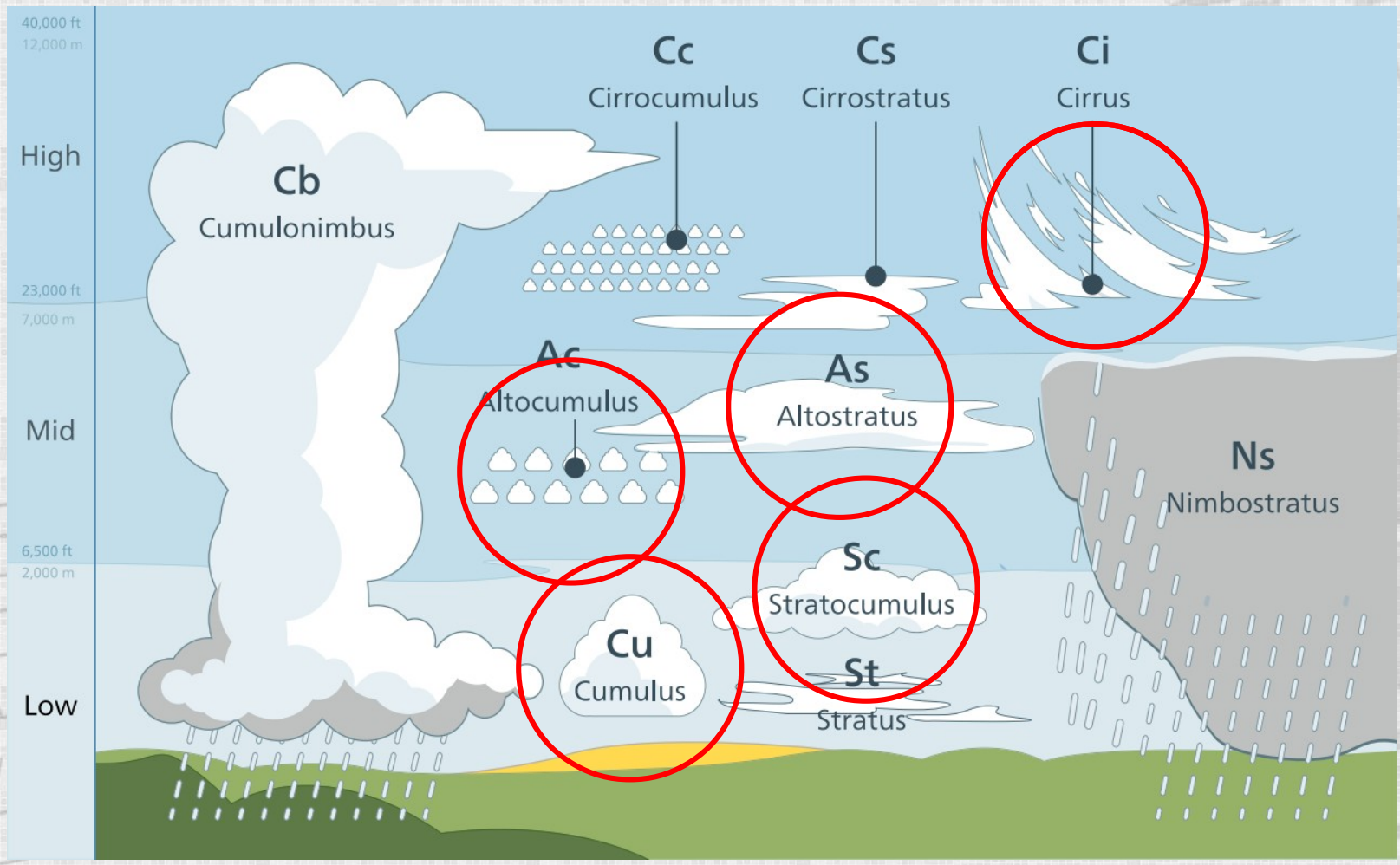
Cloud taxonomy

Clouds modelled in the current thesis:



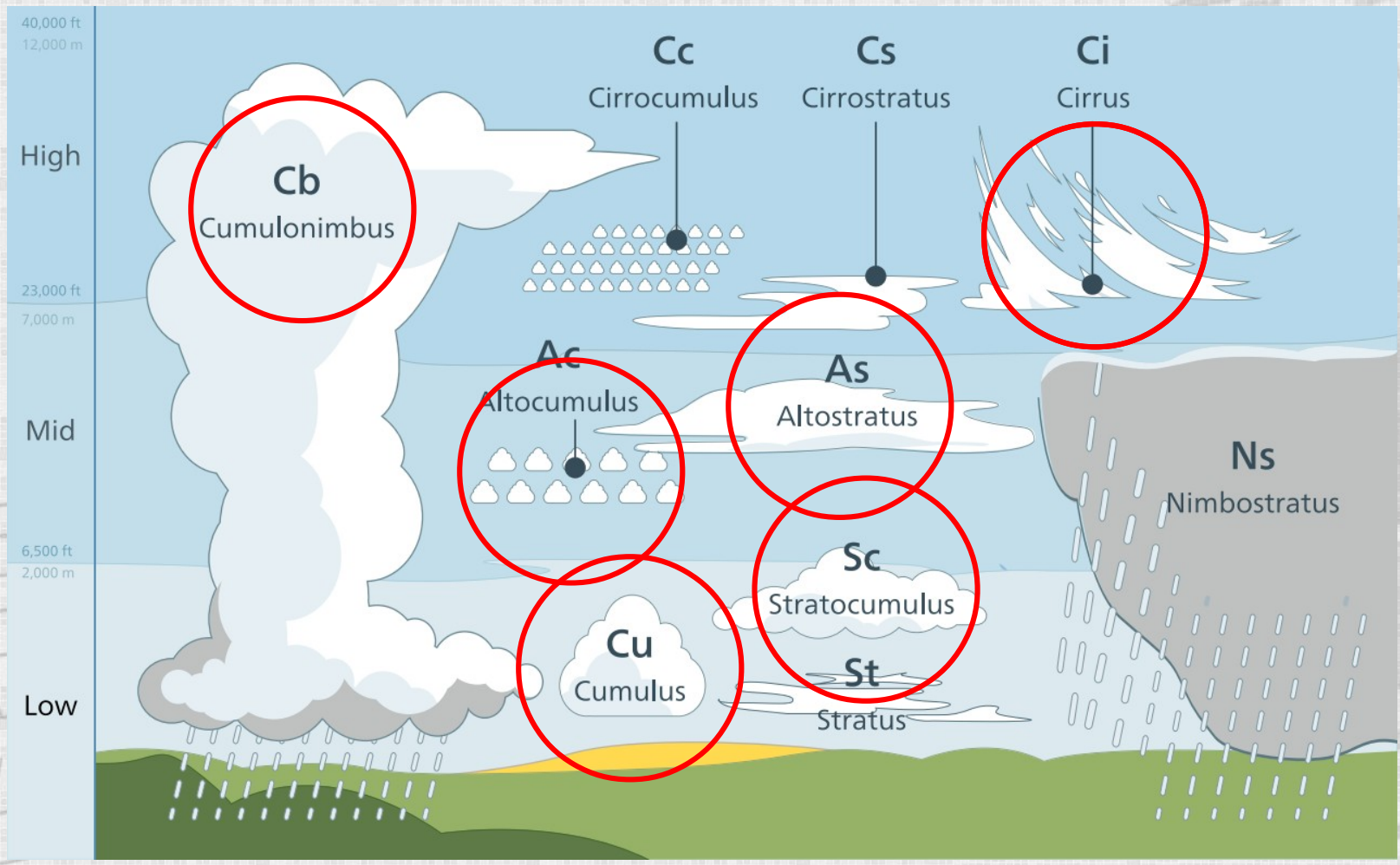
Cloud taxonomy

Clouds modelled in the current thesis:



Cloud taxonomy

Clouds modelled in the current thesis:



Radiometry

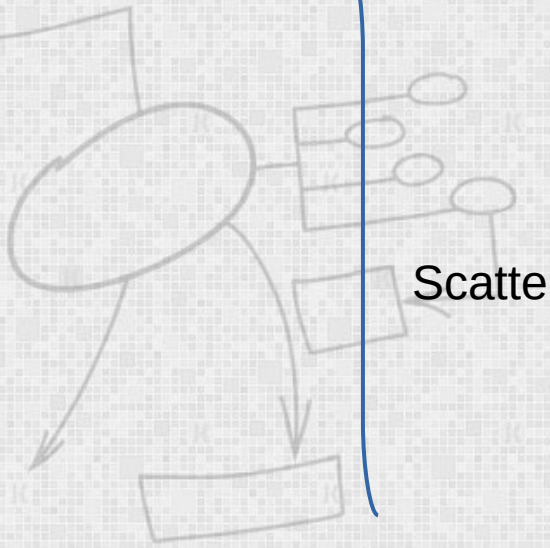
Main concepts

Absorption

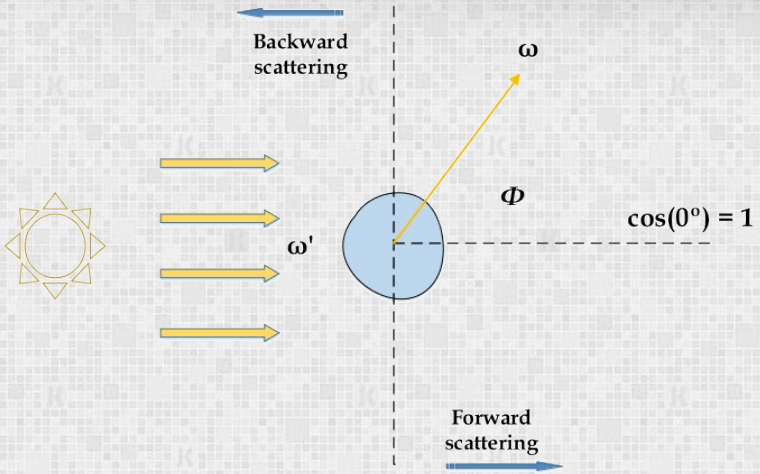
- Extinction: $K = K_s + K_a$
- Optical depth: $\tau(0,s) = \int_0^s K(\vec{x}+t\vec{\omega})dt$
- Transmittance: $T(s,s') = e^{-\tau(s,s')}$
- Absorbance: $A = -log_{10}\left(\frac{I_{output}}{I_{input}}\right) = -log_{10}T$

Scattering

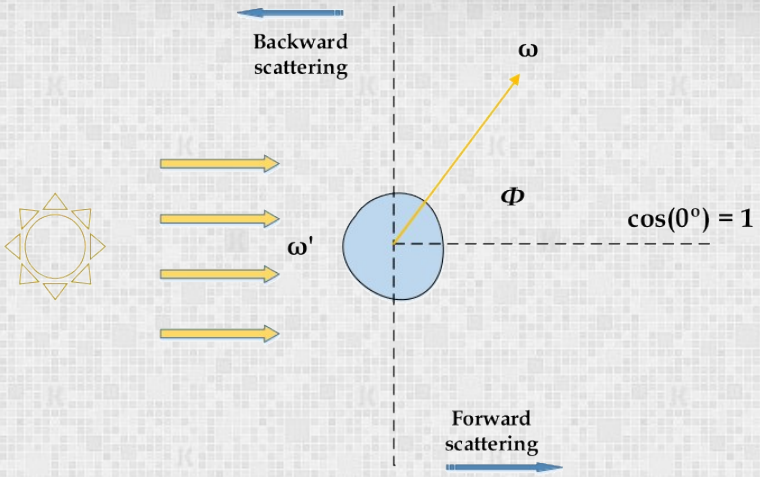
- Single scattering
- Multiple scattering
- Scattering albedo: $\sigma = \frac{K_s}{K}$



Radiometry



Radiometry

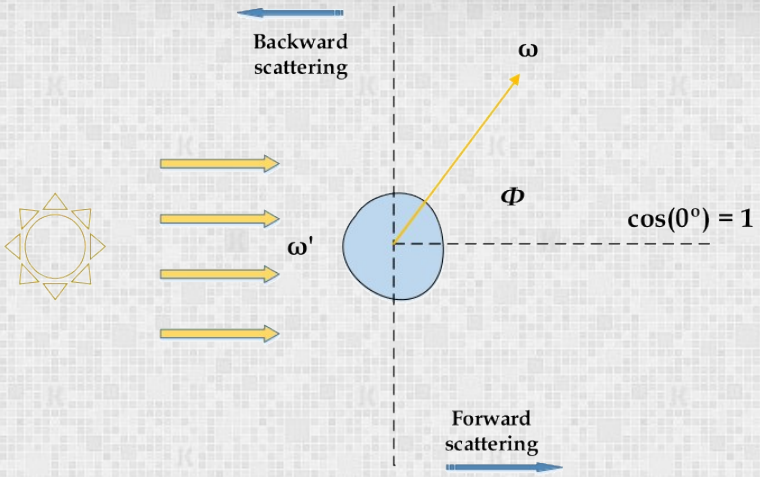


Henyey-Greenstein phase function

$$P(\theta) = \frac{1}{4\pi} \frac{1-g^2}{[1+g^2-2g\cos(\theta)]^{3/2}}$$

g { < 0 backscattering

Radiometry

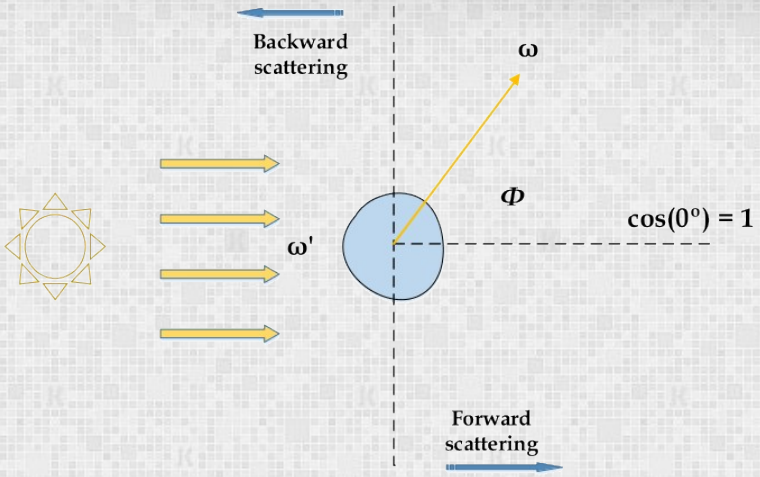


Henyey-Greenstein phase function

$$P(\theta) = \frac{1}{4\pi} \frac{1-g^2}{[1+g^2-2g\cos(\theta)]^{3/2}}$$

g
- < 0 backscattering
- = 0 isotropic scattering

Radiometry

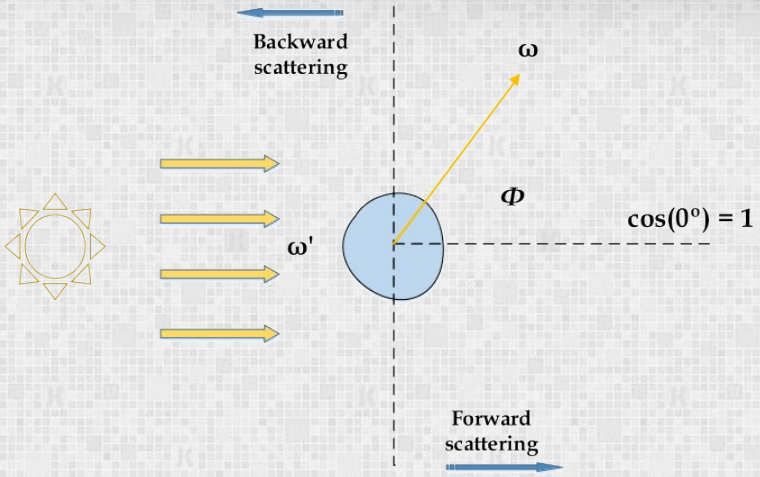


Henyey-Greenstein phase function

$$P(\theta) = \frac{1}{4\pi} \frac{1-g^2}{[1+g^2-2g\cos(\theta)]^{3/2}}$$

g
- < 0 backscattering
- = 0 isotropic scattering
- > 0 forward scattering

Radiometry



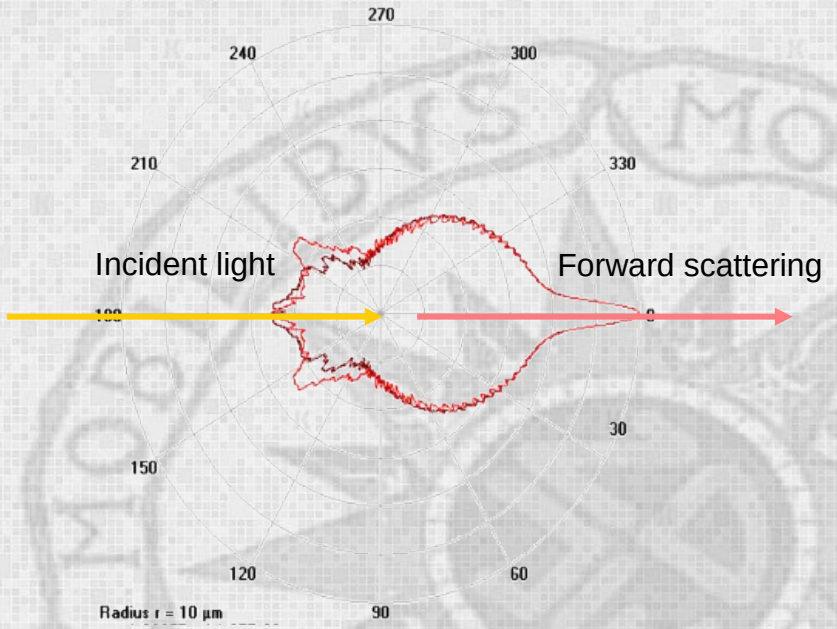
Henyey-Greenstein phase function

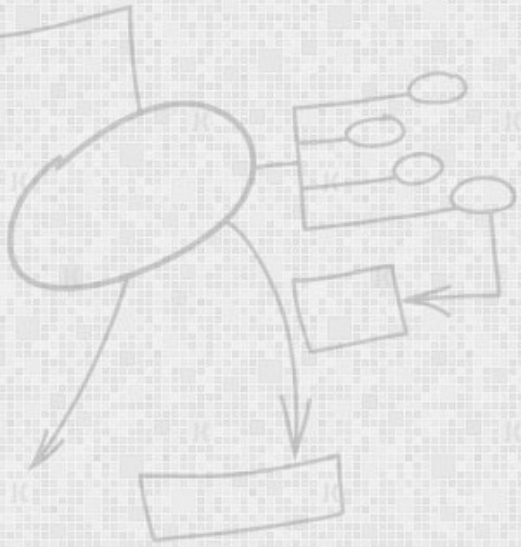
$$P(\theta) = \frac{1}{4\pi} \frac{1-g^2}{[1+g^2-2g\cos(\theta)]^{3/2}}$$

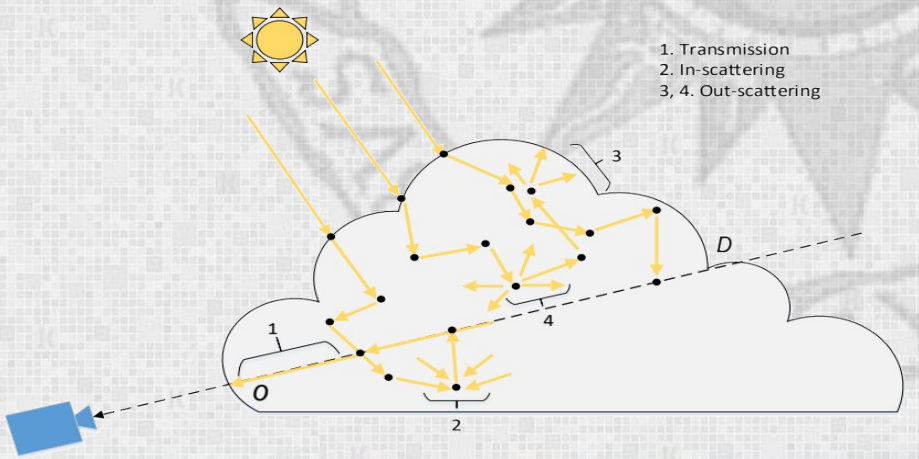
g
- < 0 backscattering
- = 0 isotropic scattering
- > 0 forward scattering

The importance of scattering

- Water droplets act as small lenses refracting light in all directions, also known as scattering:
  - 50% of light follows the direction of incidence
  - 50% of light departs in all other directions

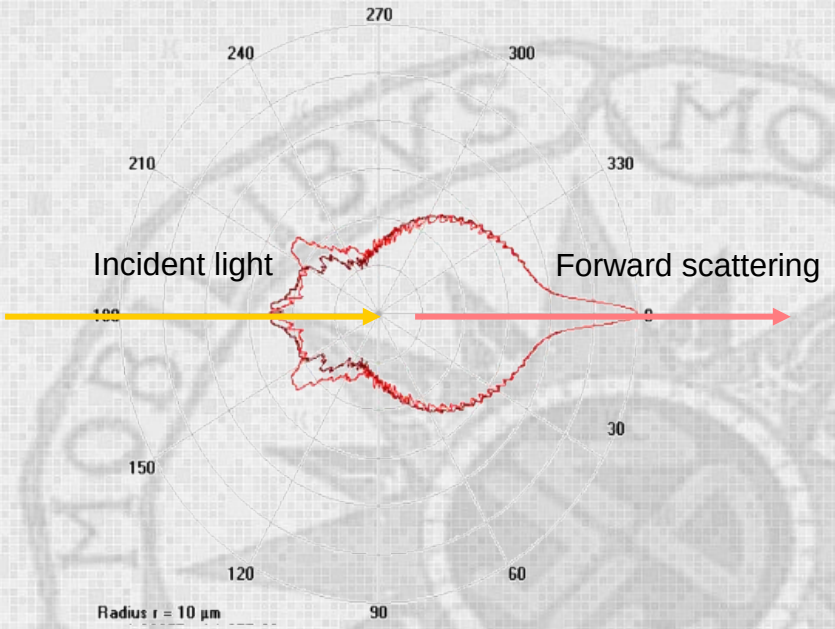


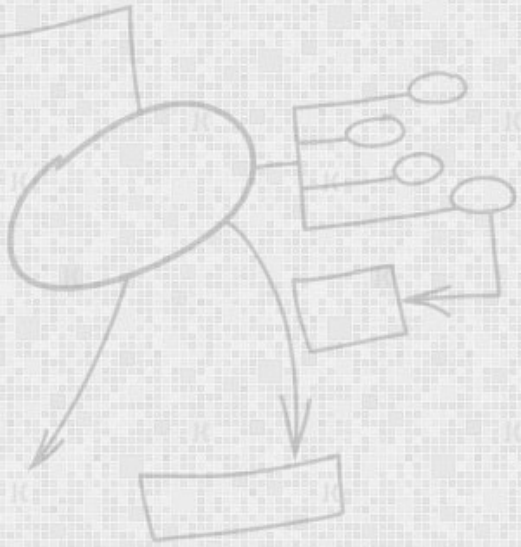


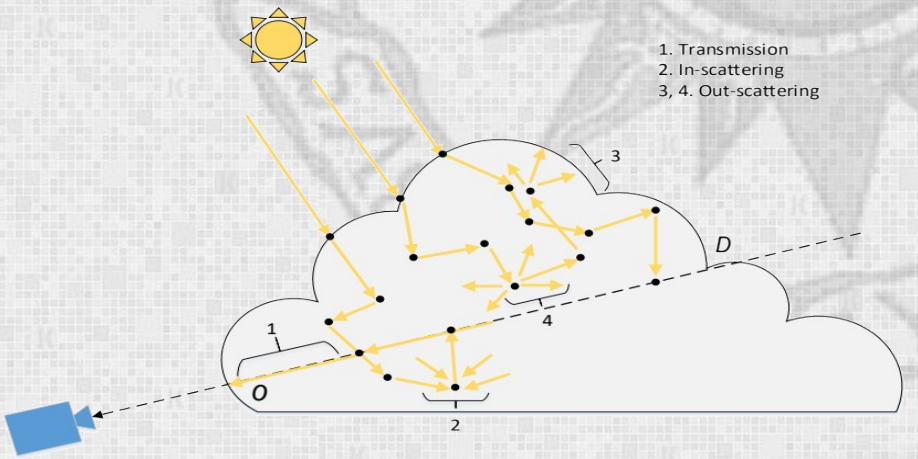


The importance of scattering

- Water droplets act as small lenses refracting light in all directions, also known as scattering:
  - 50% of light follows the direction of incidence
  - 50% of light departs in all other directions
- This phenomenon causes the cloud to look bright even in places that should be shadowed

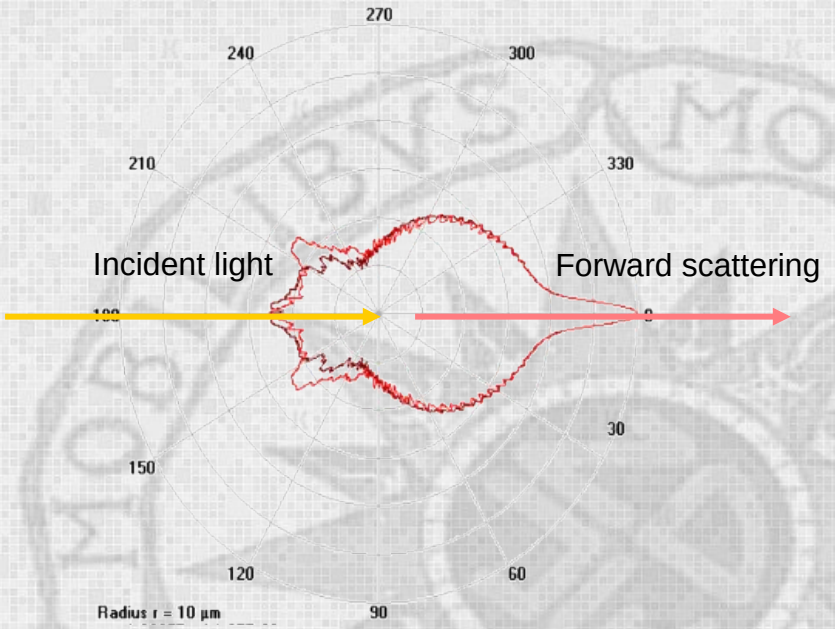


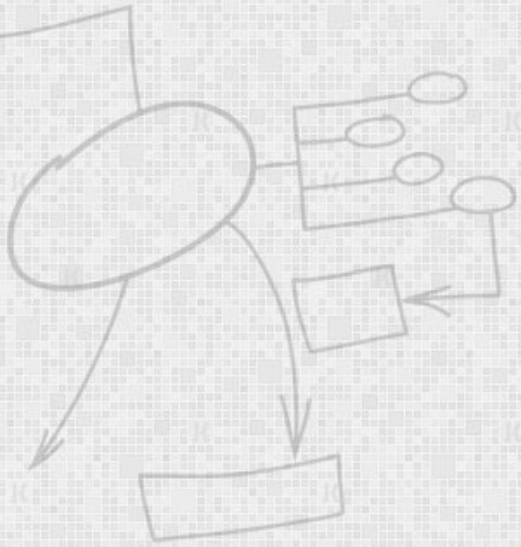


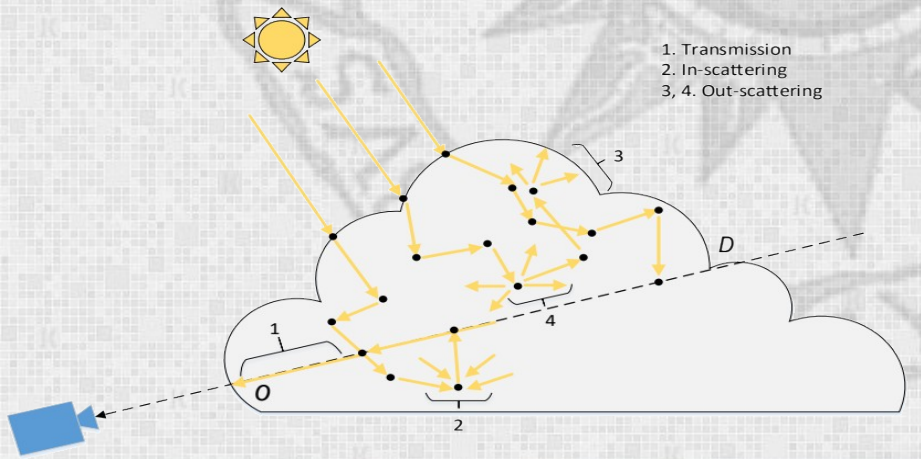


The importance of scattering

- Water droplets act as small lenses refracting light in all directions, also known as scattering:
  - 50% of light follows the direction of incidence
  - 50% of light departs in all other directions
- This phenomenon causes the cloud to look bright even in places that should be shadowed
- The phase function reproduces this assimmetry at a macroscopic level

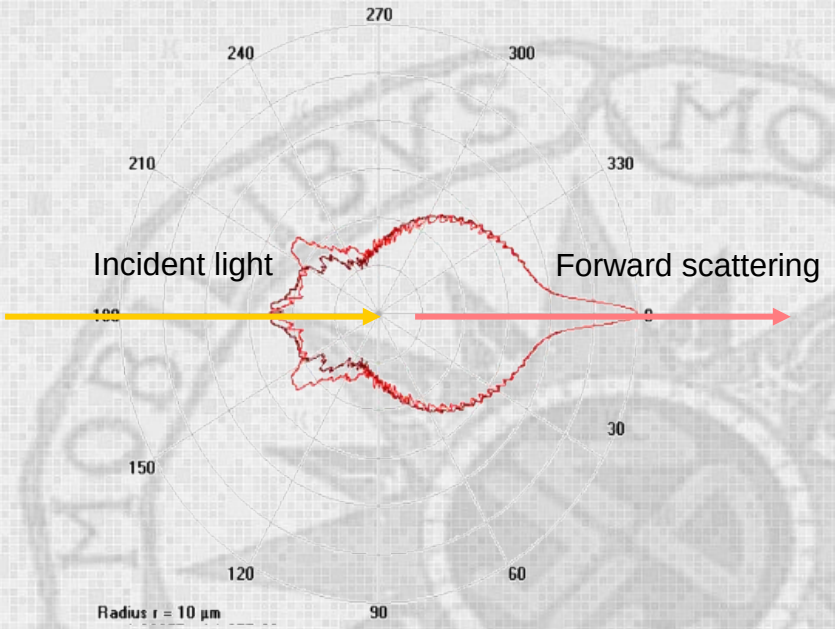


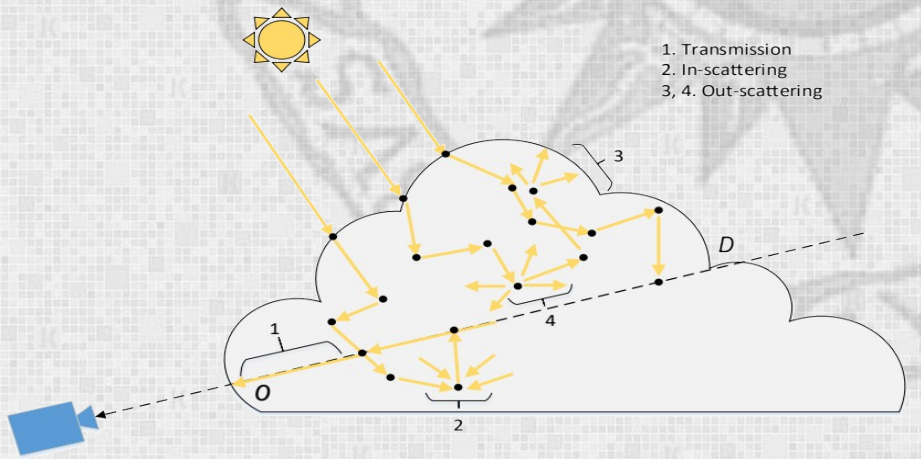




The importance of scattering

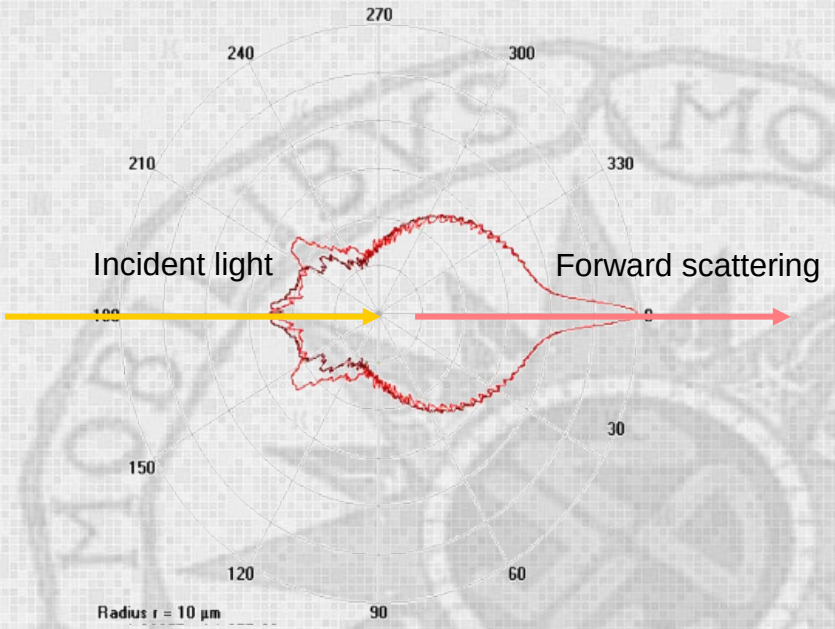
- Water droplets act as small lenses refracting light in all directions, also known as scattering:
  - 50% of light follows the direction of incidence
  - 50% of light departs in all other directions
- This phenomenon causes the cloud to look bright even in places that should be shadowed
- The phase function reproduces this assimmetry at a macroscopic level
- Computing scattering for every point recursively is computationally expensive

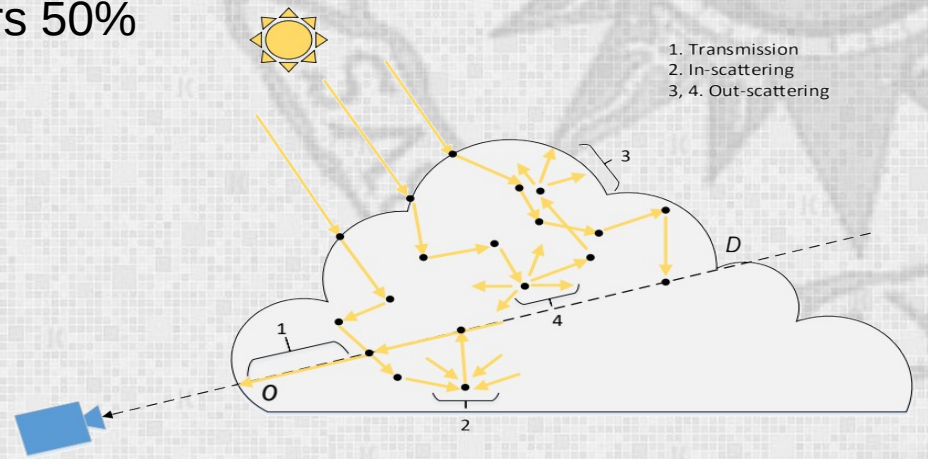




The importance of scattering

- Water droplets act as small lenses refracting light in all directions, also known as scattering:
  - 50% of light follows the direction of incidence
  - 50% of light departs in all other directions
- This phenomenon causes the cloud to look bright even in places that should be shadowed
- The phase function reproduces this assimmetry at a macroscopic level
- Computing scattering for every point recursively is computationally expensive
- Forward scattering can be computed with little extra cost during ray marching and covers 50% of the light





Light transport model

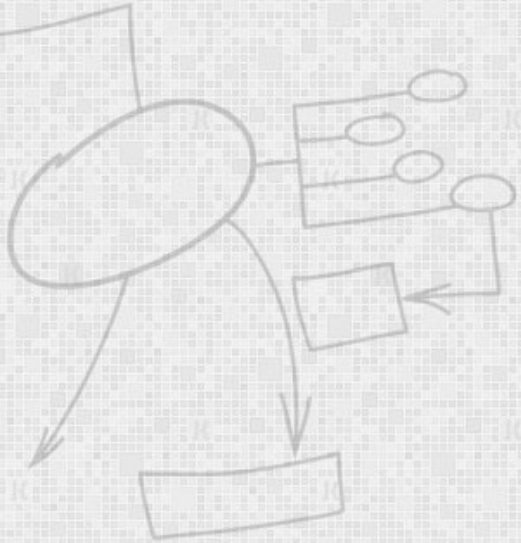
Based on the work of Nelson Max [Max95] and Mark Harris [Har02] but applied to voxelized volumetric rendering

Absorption

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_a(\vec{x})L(\vec{x},\omega) \quad \textbf{(1.9)}$$

Outscattering

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_s(\vec{x})L(\vec{x},\omega) \quad \textbf{(1.10)}$$



Light transport model

Based on the work of Nelson Max [Max95] and Mark Harris [Har02] but applied to voxelized volumetric rendering

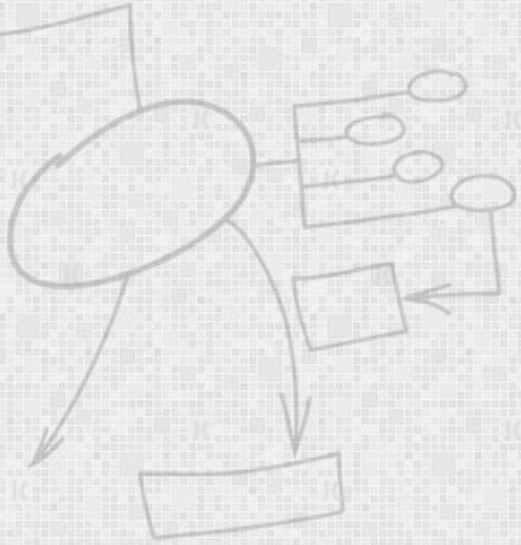
Absorption

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_a(\vec{x})L(\vec{x},\omega) \quad \textbf{(1.9)}$$

Outscattering

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_s(\vec{x})L(\vec{x},\omega) \quad \textbf{(1.10)}$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\omega) \quad \textbf{(1.11)}$$



Light transport model

Based on the work of Nelson Max [Max95] and Mark Harris [Har02] but applied to voxelized volumetric rendering

Absorption

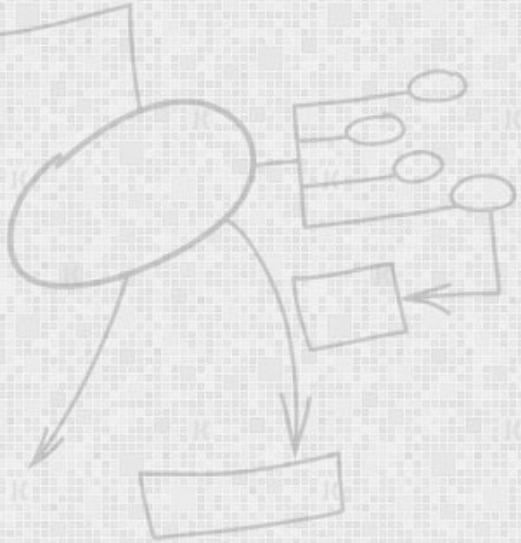
$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_a(\vec{x})L(\vec{x},\omega) \quad (1.9)$$

Outscattering

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_s(\vec{x})L(\vec{x},\omega) \quad (1.10)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\omega) \quad (1.11)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.12)$$



Light transport model

Based on the work of Nelson Max [Max95] and Mark Harris [Har02] but applied to voxelized volumetric rendering

Absorption

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_a(\vec{x})L(\vec{x},\omega) \quad (1.9)$$

Outscattering

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_s(\vec{x})L(\vec{x},\omega) \quad (1.10)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\omega) \quad (1.11)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.12)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\vec{\omega}) + K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.13)$$

Light transport model

Based on the work of Nelson Max [Max95] and Mark Harris [Har02] but applied to voxelized volumetric rendering

Absorption

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_a(\vec{x})L(\vec{x},\omega) \quad (1.9)$$

Outscattering

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_s(\vec{x})L(\vec{x},\omega) \quad (1.10)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\omega) \quad (1.11)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.12)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\vec{\omega}) + K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.13)$$

$$L(D,\vec{\omega}) = \underbrace{L(0,\vec{\omega})T(0,D)}_{\text{transmission}} + \underbrace{\int_0^D g(s)T(s,D)ds}_{\text{in-scattering}} \quad (1.14)$$

Light transport model

Based on the work of Nelson Max [Max95] and Mark Harris [Har02] but applied to voxelized volumetric rendering

Absorption

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_a(\vec{x})L(\vec{x},\omega) \quad (1.9)$$

Outscattering

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_s(\vec{x})L(\vec{x},\omega) \quad (1.10)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\omega) \quad (1.11)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.12)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\vec{\omega}) + K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.13)$$

$$L(D,\vec{\omega}) = \underbrace{L(0,\vec{\omega})T(0,D)}_{\text{transmission}} + \underbrace{\int_0^D g(s)T(s,D)ds}_{\text{in-scattering}} \quad (1.14)$$

$$g(s) = K_s(\vec{x}(s))\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x}(s),\omega')d\omega' \quad (1.15)$$

Light transport model

Based on the work of Nelson Max [Max95] and Mark Harris [Har02] but applied to voxelized volumetric rendering

Absorption

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_a(\vec{x})L(\vec{x},\omega) \quad (1.9)$$

Outscattering

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K_s(\vec{x})L(\vec{x},\omega) \quad (1.10)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\omega) \quad (1.11)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.12)$$

$$\frac{dL(\vec{x},\vec{\omega})}{ds} = -K(\vec{x})L(\vec{x},\vec{\omega}) + K_s(\vec{x})\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x},\omega')d\omega' \quad (1.13)$$

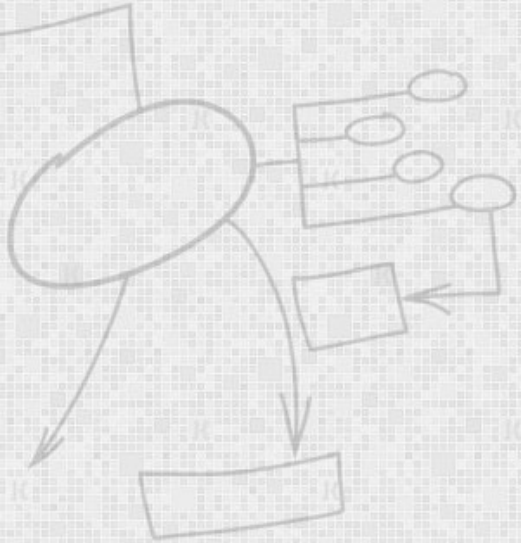
$$L(D,\vec{\omega}) = \underbrace{L(0,\vec{\omega})T(0,D)}_{\text{transmission}} + \underbrace{\int_0^D g(s)T(s,D)ds}_{\text{in-scattering}} \quad (1.14)$$

$$g(s) = K_s(\vec{x}(s))\int_{4\pi} P(\vec{x},\vec{\omega},\vec{\omega}')L(\vec{x}(s),\omega')d\omega' \quad (1.15)$$

State of the art

Ontogenetics vs. Physically based models

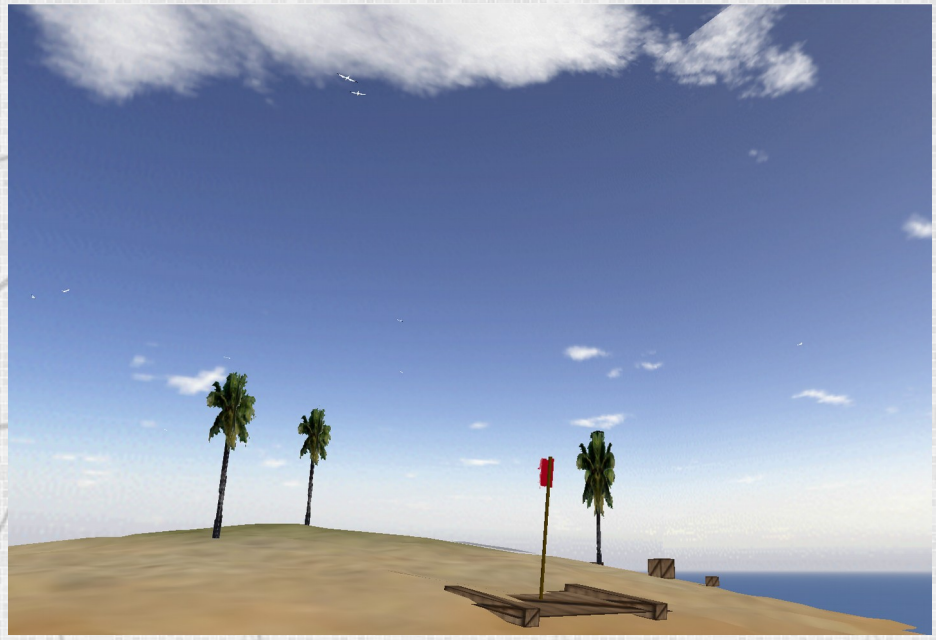
1. **Texturized primitives**
2. **Particle systems**
3. **Geometry distortion**
4. **Volumetric rendering**



State of the art

Ontogenetics vs. Physically based models

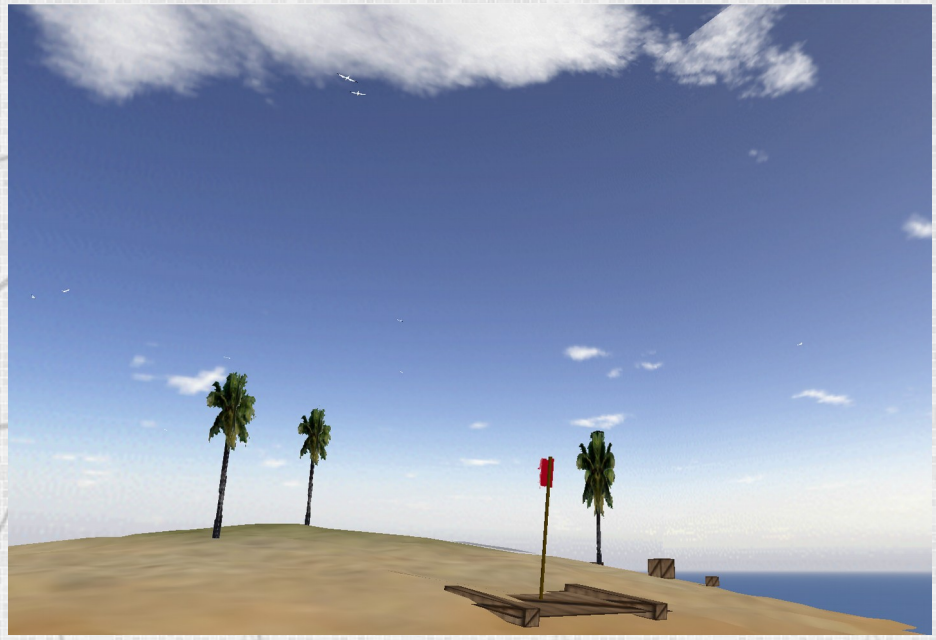
Texturized primitives



State of the art

Ontogenetics vs. Physically based models

Texturized primitives

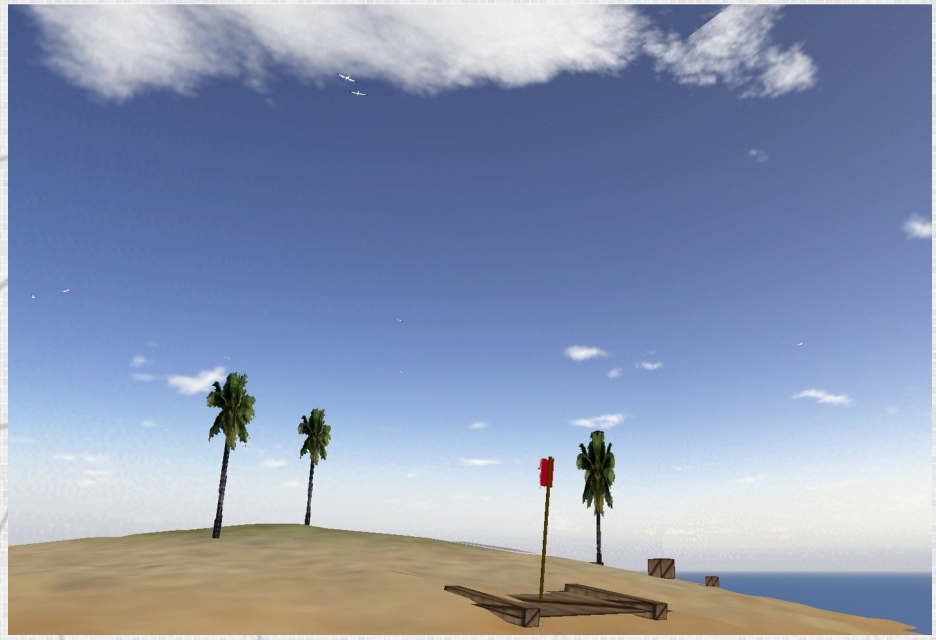


Author's contributions

State of the art

Ontogenetics vs. Physically based models

Texturized primitives

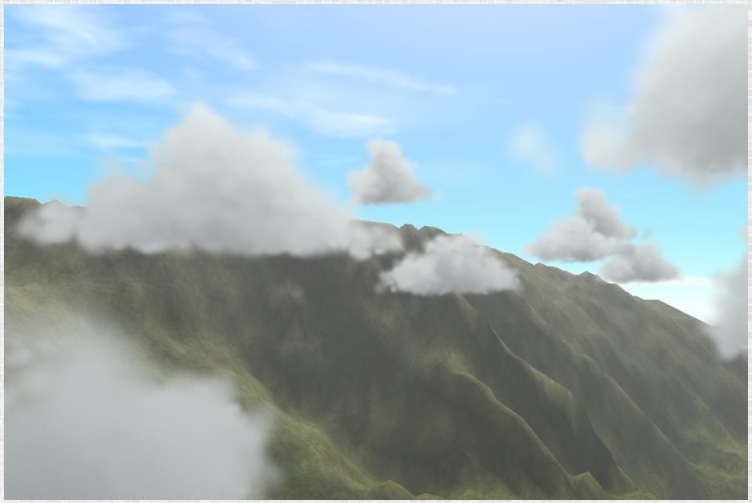


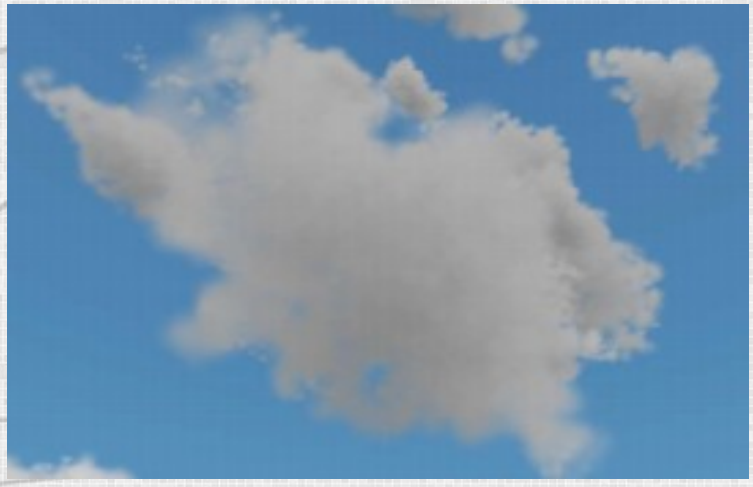
Author's contributions

- Approaching, manoeuvering around, traversing the gaseous form
- Cloud animation
- Dynamic lighting in real-time
- More realism

State of the art

Particle systems

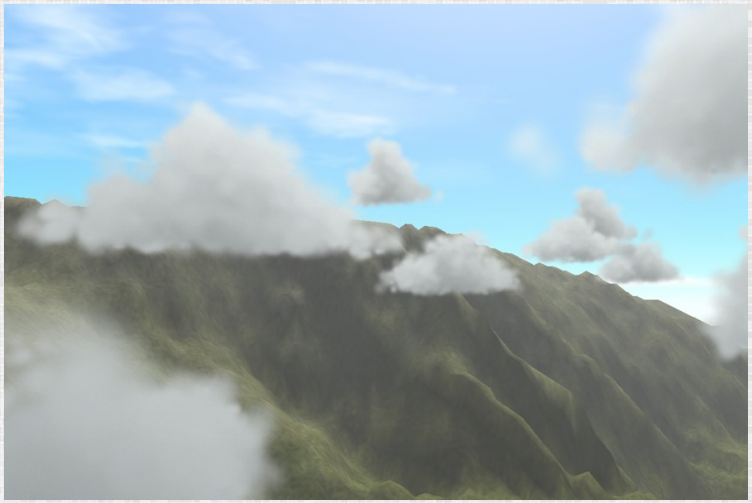


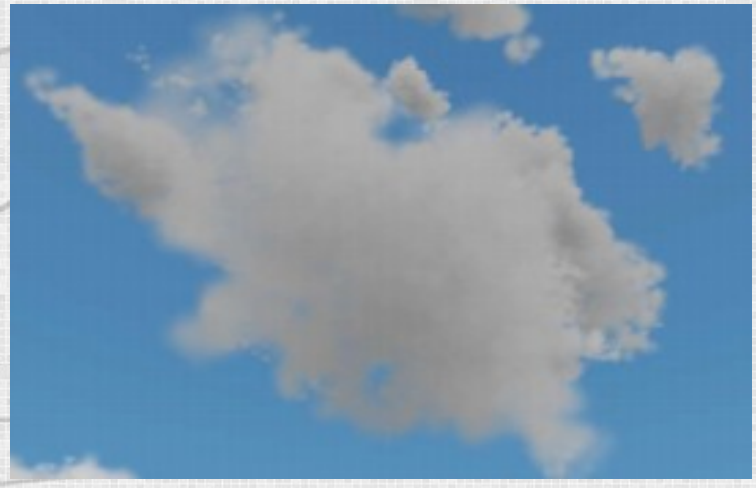


Author's contributions

State of the art

Particle systems



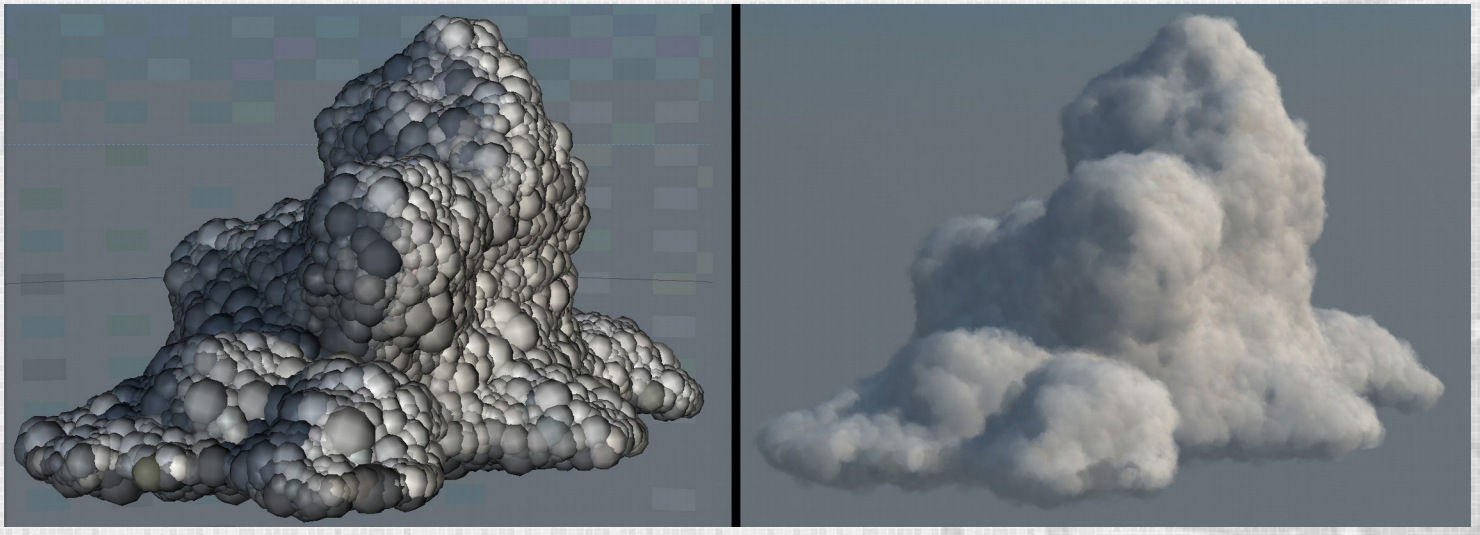


Author's contributions

- Use of tens of pseudospheroids instead of thousands of particles
- More gaseous form realism thanks to the volumetric rendering
- Realistic time-evolving cloud animation

State of the art

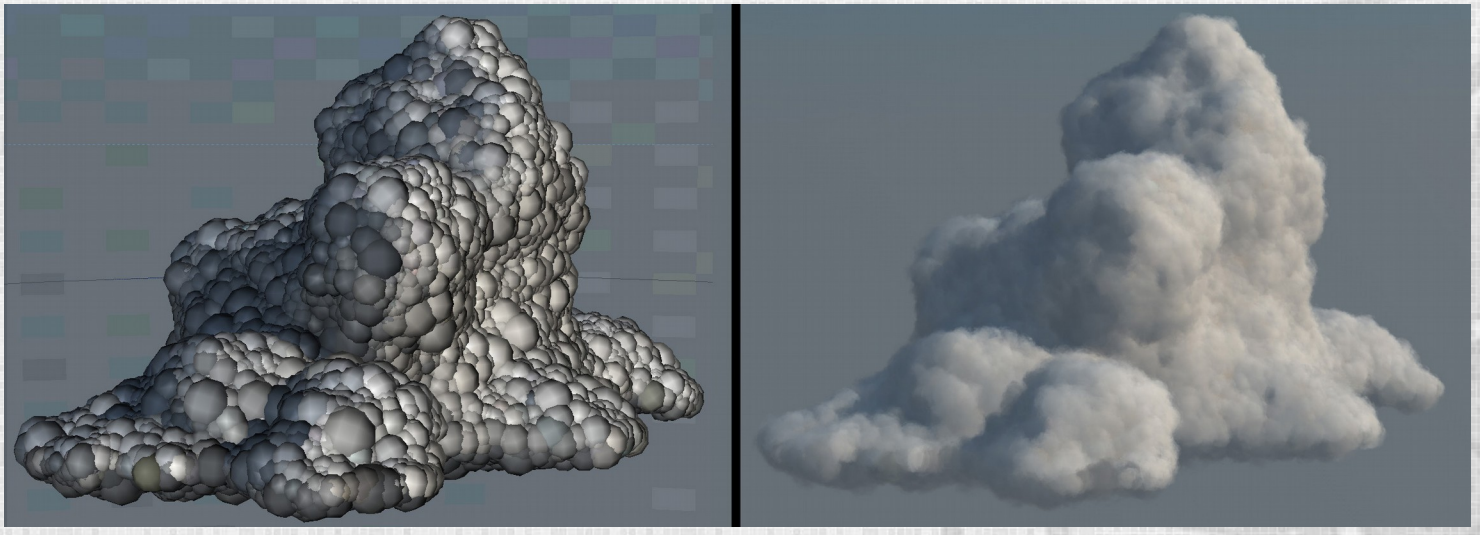
Geometry distortion



Author's contributions

State of the art

Geometry distortion

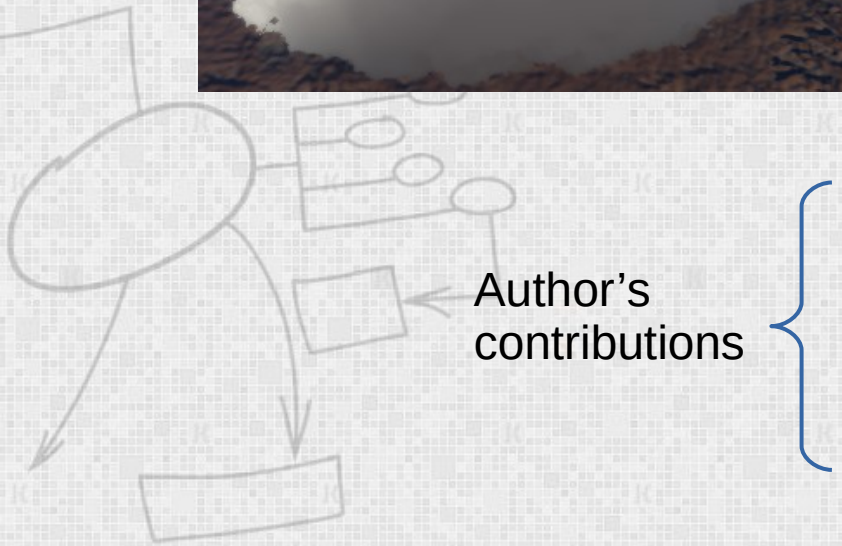


Author's contributions

- Arbitrary shaped clouds
- Better lighting systems than Phong or Gouraud
- Improved cloud dynamics and animation

State of the art

Volumetric rendering



Author's contributions

State of the art

Volumetric rendering

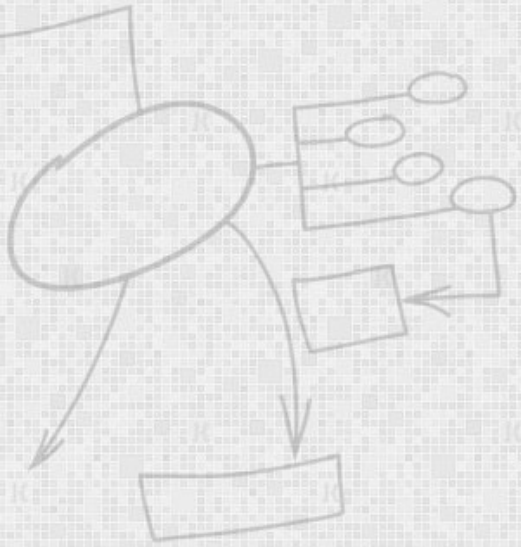
Author's contributions

- Improved performance
- Real-time cloud dynamics, animation and morphing

Pros and cons

	Texturized primitives	Particle systems	Geometry distortion	Volumetric rendering
Procedural	✓	✗	✗	✓
Animation	✗	✓	✓	✓
Traverse	✗	✓	✗	✓
Interactive	✓	✓	✓	✓
Manoeuvring around	✗	✓	✓	✓
All kinds of clouds	✓	✗	✗	✓
Realism	✓	✗	✓	✓
Performance	✓	✓	✓	✗
State of the art	✓	✗	✗	✓

Table 1.1 Features per method



Pros and cons

	Texturized primitives	Particle systems	Geometry distortion	Volumetric rendering
Procedural	✓	✗	✗	✓
Animation	✗	✓	✓	✓
Traverse	✗	✓	✗	✓
Interactive	✓	✓	✓	✓
Manoeuvring around	✗	✓	✓	✓
All kinds of clouds	✓	✗	✗	✓
Realism	✓	✗	✓	✓
Performance	✓	✓	✓	✗
State of the art	✓	✗	✗	✓

Table 1.1 Features per method

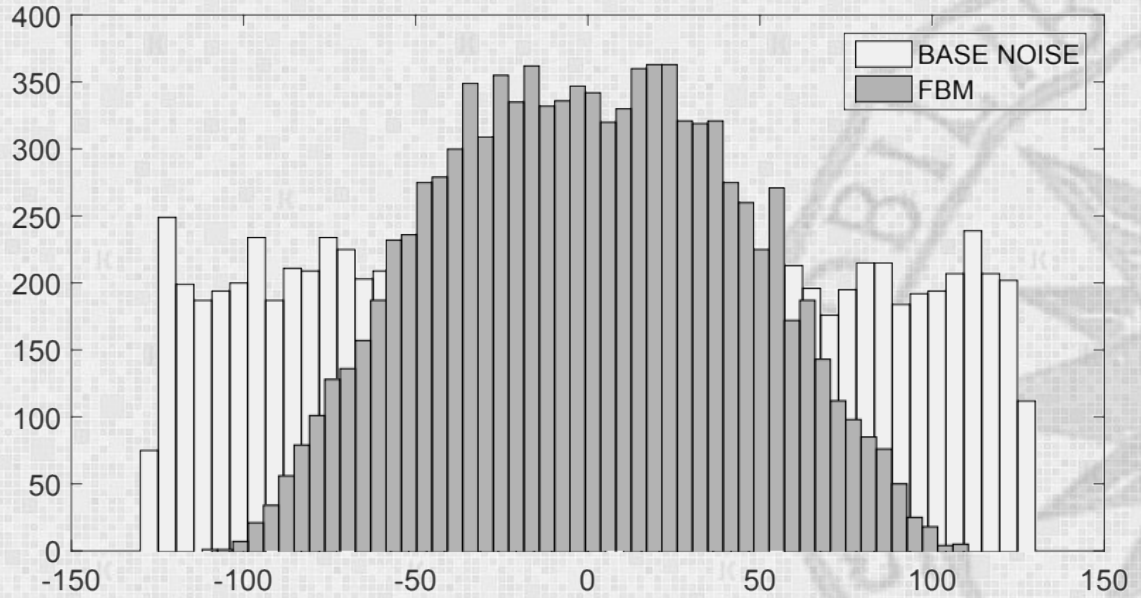
	Mukina et al. [MB15]	Harris [Har03]	Kniss et al.[Kni+02]	Kallweit et al. [Kal+17]	de Parga [JG18]
Procedural	✓	✗	✗	✓	✓
Animation	✗	✓	✓	✗	✓
Traverse	✗	✓	✗	✗	✓
Interactive	✓	✓	✓	✓	✓
Manoeuvring around	✗	✓	✓	✓	✓
All kinds of clouds	✓	✗	✗	✗	✓
Realism	✓	✗	✓	✓	✓
Performance	✓	✓	✓	✗	✓
State of the art	✓	✗	✗	✓	✓

Table 1.2 Features per author

PART II

Texture generation

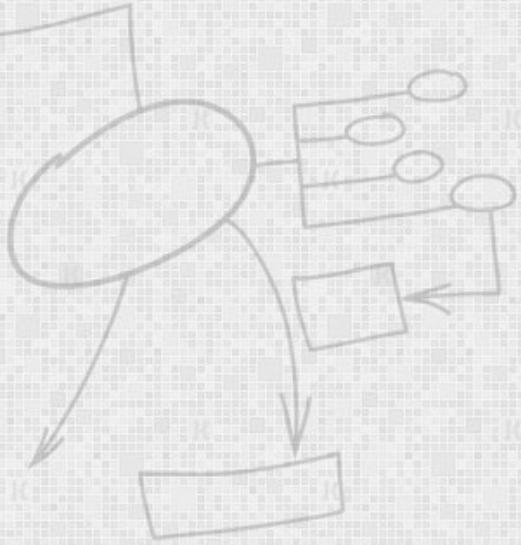
A three-dimensional cube of 64 x 64 x 64 single-precision floats is filled with uniform random noise pre-calculated in CPU before generating **fBm** (fractal Brownian motion) noise in GPU



$$fbm(x,y,z) = \sum_{i=1}^{n} w^i \cdot perlin(s^i x, s^i y, s^i z)$$

where *w* = 1/2 and *s* is 2.

Pseudospheroids



Pseudospheroids

$$\gamma \leftarrow e\left[\frac{-\|rayPos-\vec{sphereCenter}\|}{radius\times\left(\left(1-\kappa\right)+2\kappa fbm(x,y,z)\right)}\right]$$

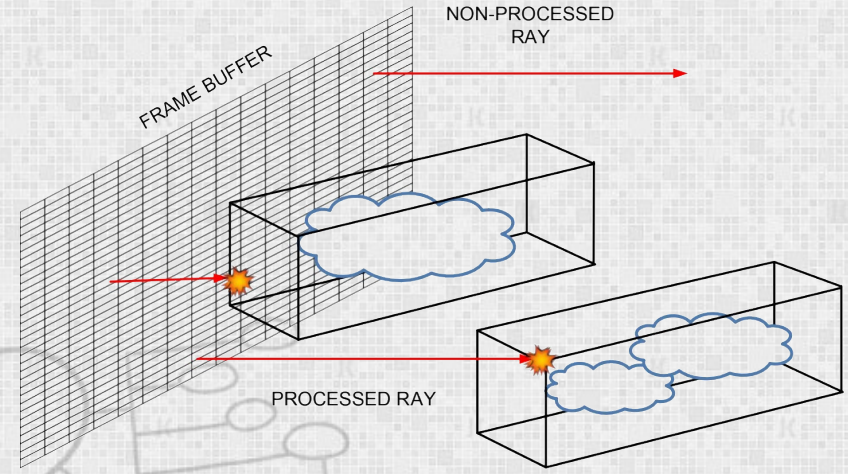
Basic volumetric cloud rendering algorithm in GPU

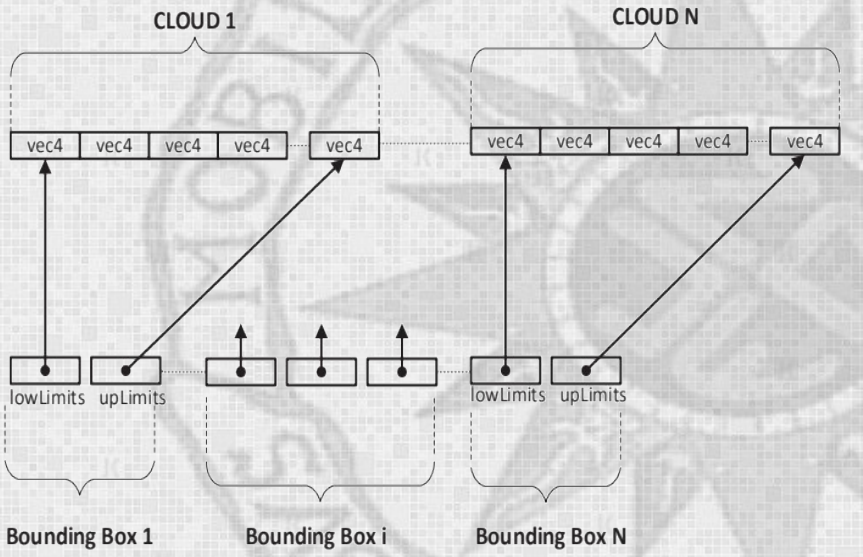
```
1  Function getCandidates(rayOrigin, rayDirection, i)
2      for j ← 0 < number of spheres in boundingbox[i] do
3          temp ← rayOrigin − sphereCenter
4          a ← rayDirection · rayDirection {Dot products}
5          b ← 2.0 · rayDirection · temp
6          c ← temp · temp − radius²
7          Δ ← b² − 4ac
8          if Δ > 0 then
9              σ ← √Δ {There is a collision}
10             λ_in ← (−b−σ)/(2.0×a)
11             λ_out ← (−b+σ)/(2.0×a)
12             candidates[k] ← (λ_in, λ_out, j, i)
13             k = k+1
14         end
15     end
16     return (candidates,k)
17 Function sortCandidates (candidates, n)
18     for k ← 1 < n do
19         aux = candidates[k] {Insertion-sort algorithm}
20         h = k−1
21         while ((h >= 0) and (aux_λin < candidates[h]_λin)) do
22             candidates[h + 1] = candidates[h]
23             h = h−1
24         end
25         candidates[h + 1] = aux
26     end
```

```
27 Function rayTrace(rayOrigin, rayDirection)
28     B ← boundingboxDetection(rayOrigin, rayDirection)
29     τ ← 1
30     R ← (0,0,0,0) {Consider alpha-channel}
31     for each (i in B) do
32         (C,n) ← getCandidates(rayOrigin, rayDirection, i)
33     end
34     candidates ← sortCandidates(C,n)
35     for j ← 0 < n do
36         λ ← candidates[j]_λin
37         λ_out ← candidates[j]_λout
38         while λ ≤ λ_out do
39             rayPos ← rayOrigin + λ · rayDirection
40             ρ ← fbm(rayPos) {Trace pseudospheroid}
41             γ ← e^(−‖rayPos − sphereCenter‖/(r((1−κ)+2kρ)))
42             if ρ < γ then
43                 R ← R+ lighting(ρ, rayPos, rayDir, sunDir,
44                                 voxelGrid[candidates[j]_i], sunColor)
45                 if τ < 10^−6 then
46                     break for (34:)
47                 end
48             end
49             λ ← λ + A · e^−(‖rayOrigin − cloudCenter‖ · δ) {LOD}
50         end
51     end
52     return R {Blend with the sky}
```

Bounding boxes improvement

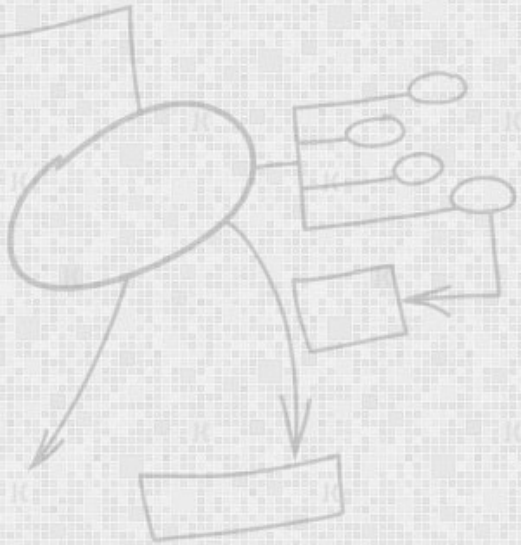
- Thanks to the Smits' [Smi02] algorithm and the improvement of A. Williams et al. [Wil+05]
- The proposed model only needs around ~35 pseudospheres to generate a cumulus





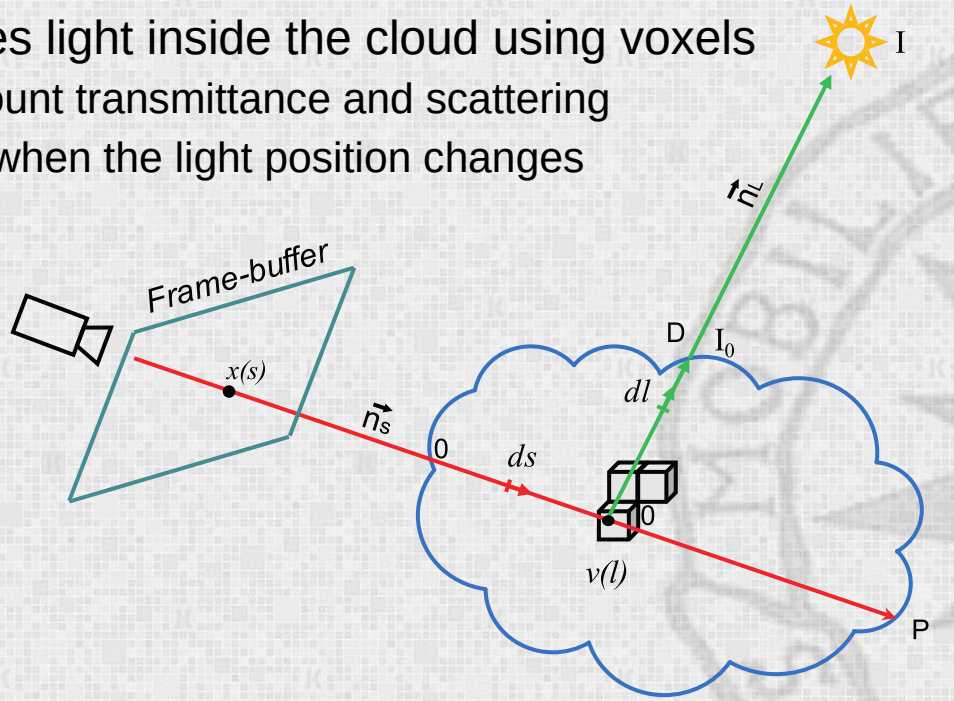
Radiometric model

- Hybrid rendering method balances workload between CPU and GPU



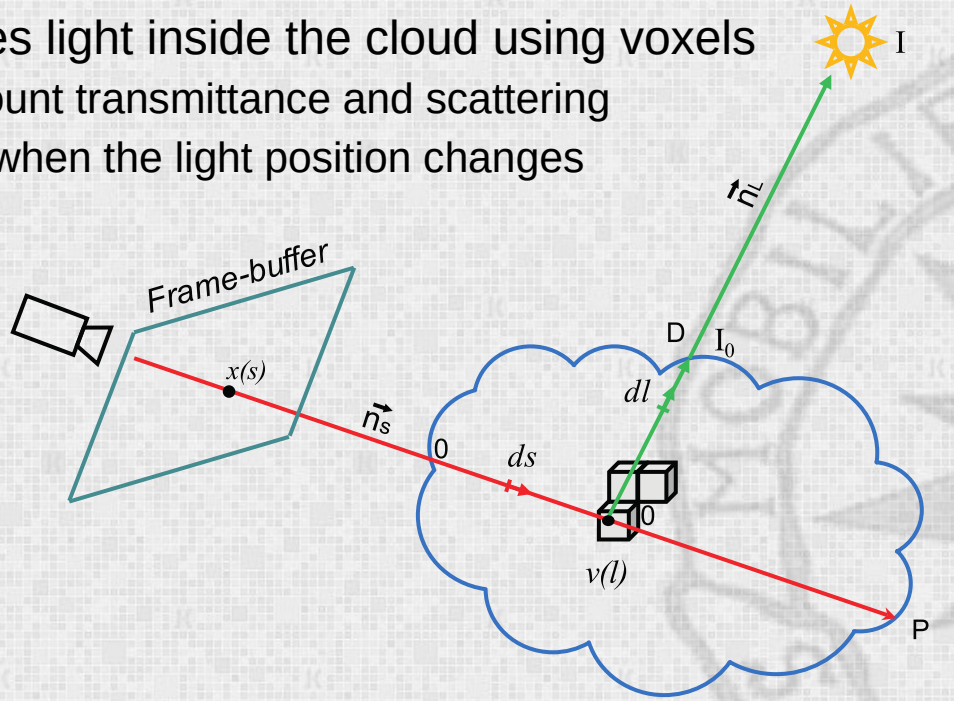
Radiometric model

- Hybrid rendering method balances workload between CPU and GPU
- CPU pre-calculates light inside the cloud using voxels
  - Takes into account transmittance and scattering
  - Executed only when the light position changes



Radiometric model

- Hybrid rendering method balances workload between CPU and GPU
- CPU pre-calculates light inside the cloud using voxels
  - Takes into account transmittance and scattering
  - Executed only when the light position changes

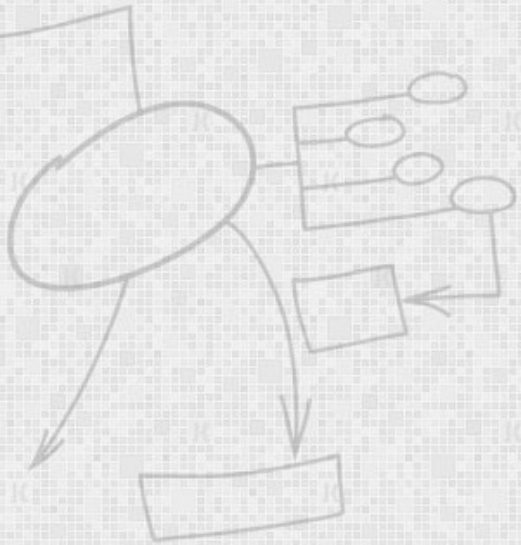


- GPU calculates projection in frame buffer
  - Takes into account cloud transparency and light emitted from the cloud (scattering)
  - Executed every frame

Radiometric equations

- Pre-computed lighting:
  - Computed for every ray from the voxel to the light source
  - First term represents light reaching the voxel inside the cloud
  - Second term represents light scattered at every point along the ray collected in the voxel and taking into account attenuation inside the cloud

$$L(v) = \underbrace{I_0 \cdot T(0,D)}_{\text{absorption}} + \underbrace{\int_0^D C(l) \cdot T(0,l)dl}_{\text{scattering}}$$



Radiometric equations

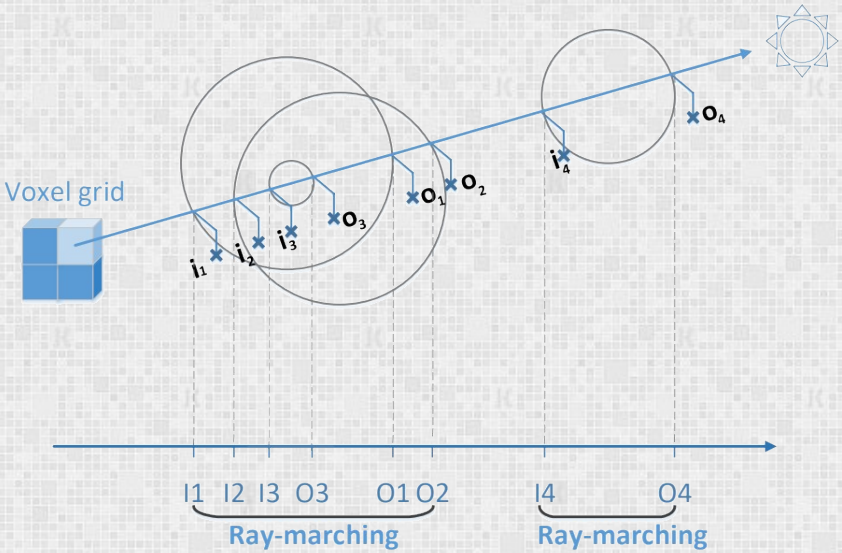
- Pre-computed lighting:
  - Computed for every ray from the voxel to the light source
  - First term represents light reaching the voxel inside the cloud
  - Second term represents light scattered at every point along the ray collected in the voxel and taking into account attenuation inside the cloud

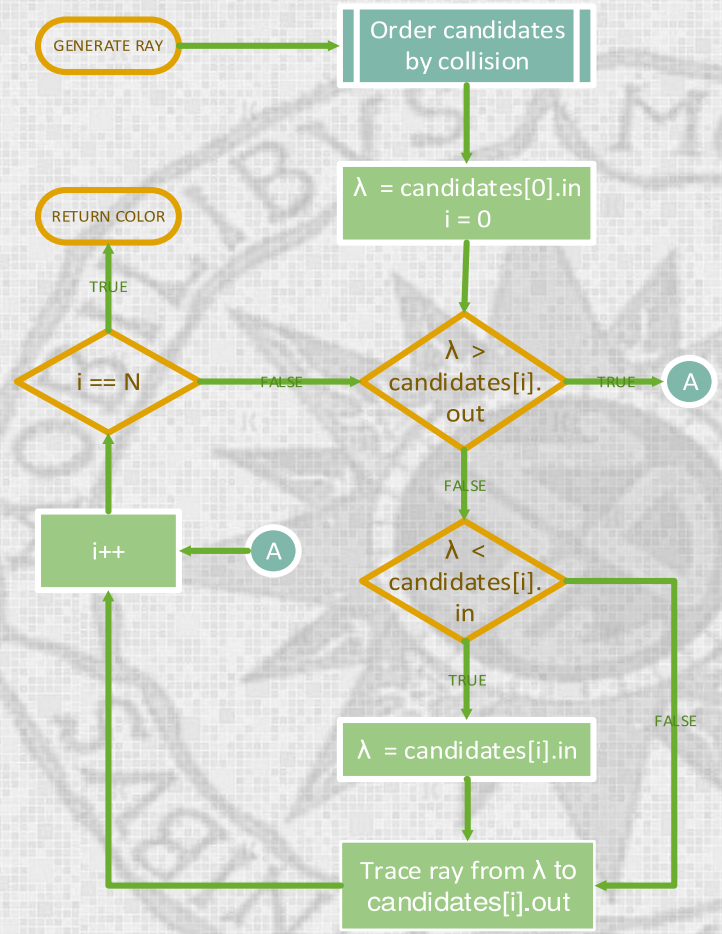
$$L(v) = \underbrace{I_0 \cdot T(0,D)}_{\text{absorption}} + \underbrace{\int_0^D C(l) \cdot T(0,l)dl}_{\text{scattering}}$$

- Real-time rendering
  - Computed for every point along the ray traversing the cloud from the camera
  - First term represents light from each voxel along the ray reflected in the gas volume according to its density
  - Second term represents light from each voxel along the ray scattered according to gas density and phase function collected in the forward direction
  - Both terms are affected by attenuation from each point to the camera

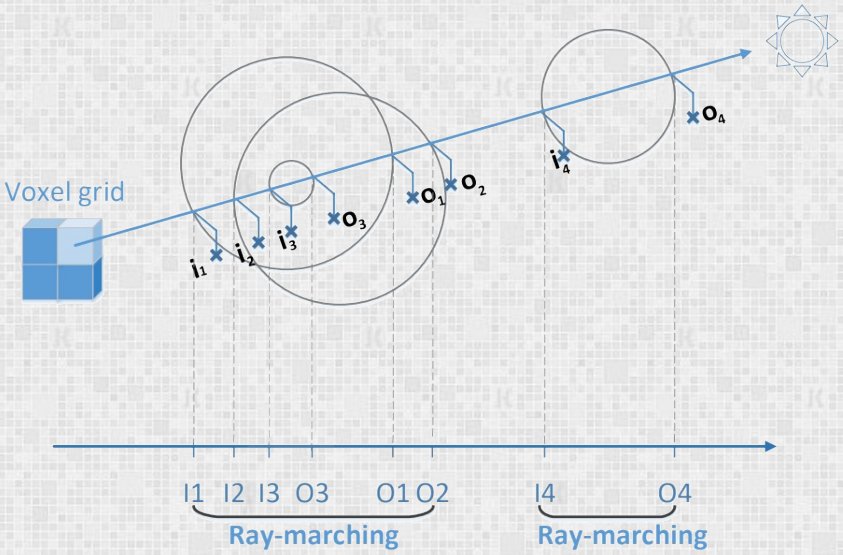
$$I(\vec{x}) = \int_0^P \left[\underbrace{L(s)\tau(s)\kappa_1}_{\text{reflection}} + \underbrace{L(s)\tau(s)P(\vec{n}_s,\vec{n}_l)\kappa_2}_{\text{scattering}} \right] \cdot T(0,s)ds$$

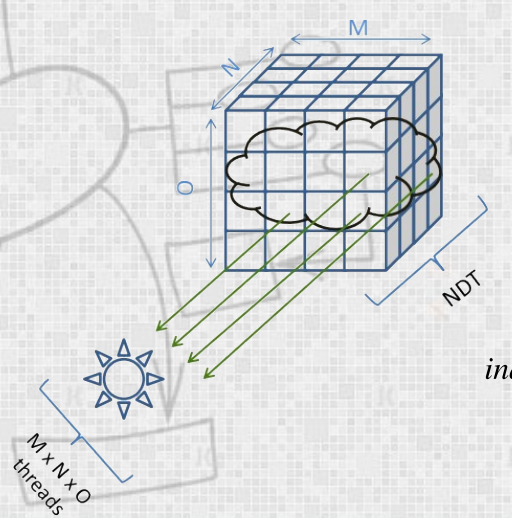
Shading improvement
(The no-duplicate-tracing algorithm NDT)





Shading improvement
(The no-duplicate-tracing algorithm NDT)

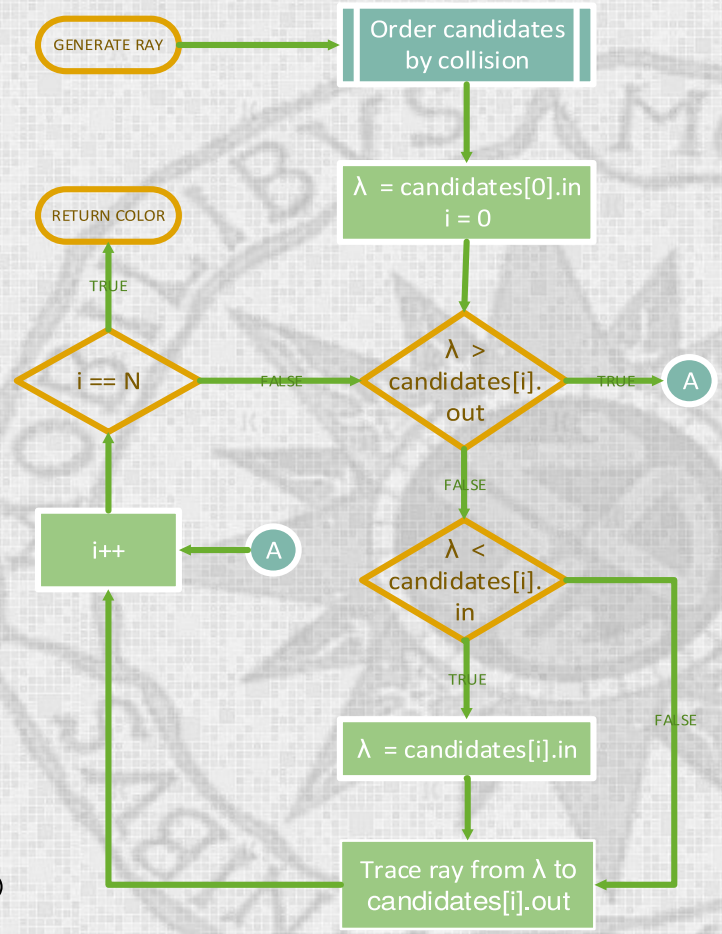




$$Block_{x,y,z} = \left\lceil \frac{Dim_{x,y,z}}{T_{x,y,z}} \right\rceil$$

```
dim3 blocks(Block_x, Block_y, Block_z)
dim3 threads(T_x, T_y, T_z)
kernel<<<blocks, threads>>>(params...)
```

$$index_{i,j,k} = blockIdx_{x,y,z} \times blockDim_{x,y,z} + threadIdx_{x,y,z}$$

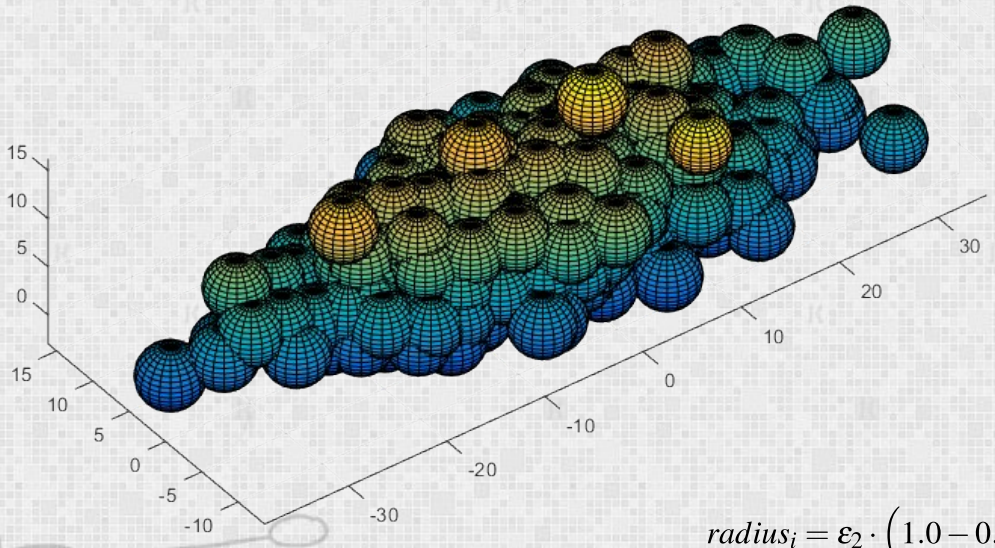


Radiometric model

Radiometric model

Radiometric model

Cloud shape improvement
(3D Gaussian cumulus)



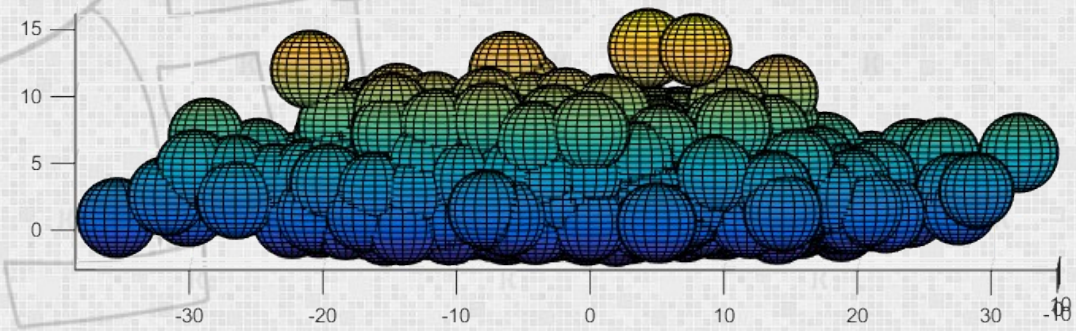
$$\begin{cases} P_x = C_x + \sim \mathcal{N}(\mu_x, \sigma_x) \\ P_y = C_y + \sim \mathcal{N}(\mu_y, \sigma_y) \\ P_z = C_z + \sim \mathcal{N}(\mu_z, \sigma_z) \end{cases} \quad (2.1)$$

Axis	μ	σ	Clamped
X	μ_x	σ_x	$[-k_1\sigma_x, k_2\sigma_x]$
Y	μ_y	σ_y	$[\mu_y, t\sigma_y]$
Z	μ_z	σ_z	$[-m_1\sigma_z, m_2\sigma_z]$

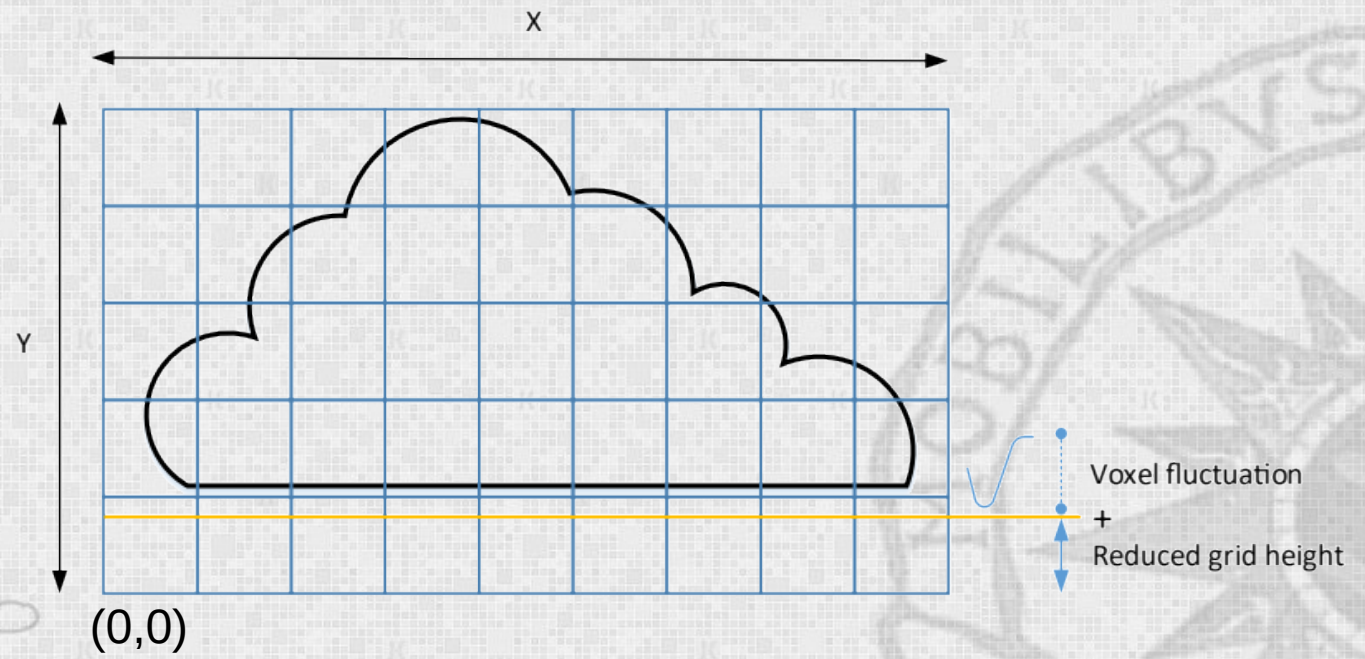
(2.2)

$$radius_i = \frac{\varepsilon_1}{|P_x - C_x| + |P_z - C_z| + 1.0} \quad (2.3)$$

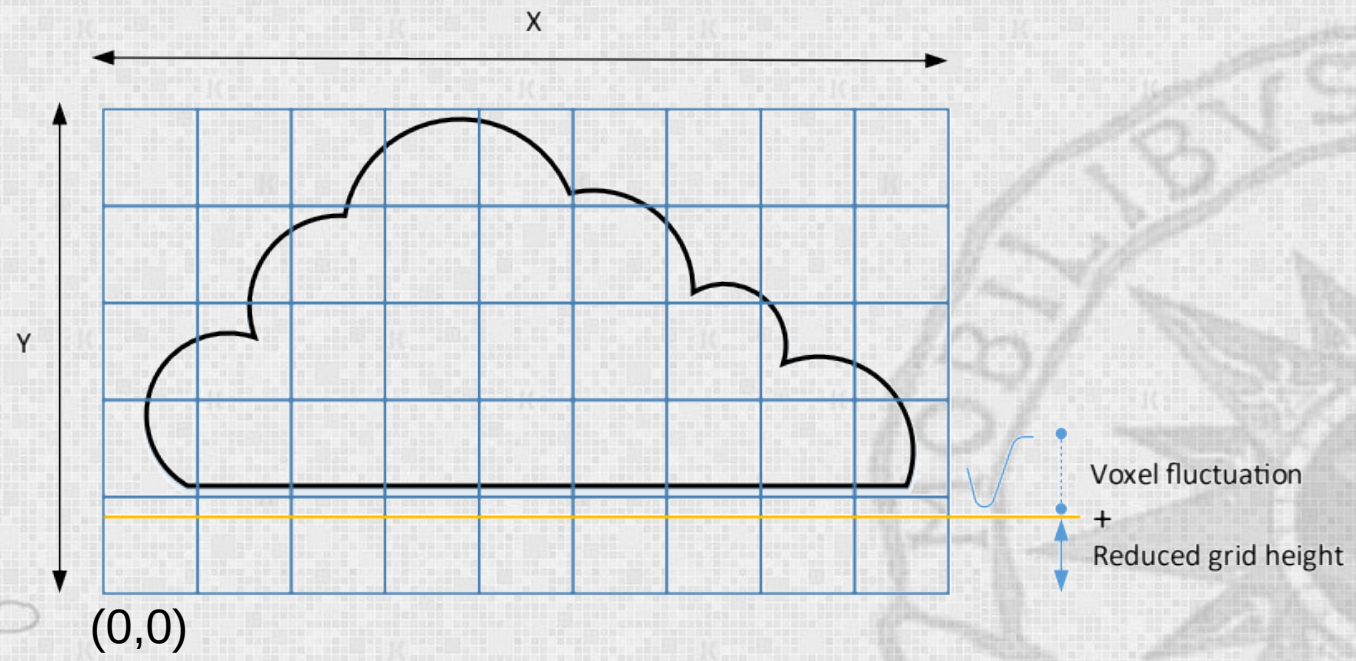
$$radius_i = \varepsilon_2 \cdot \left(1.0 - 0.1\sqrt{((P_x - C_x)/2\sigma_x)^2 \cdot ((P_y - C_y)/2\sigma_y)^2 \cdot ((P_z - C_z)/2\sigma_z)^2}\right) \quad (2.4)$$



Cloud shape improvement
(Level of condensation)

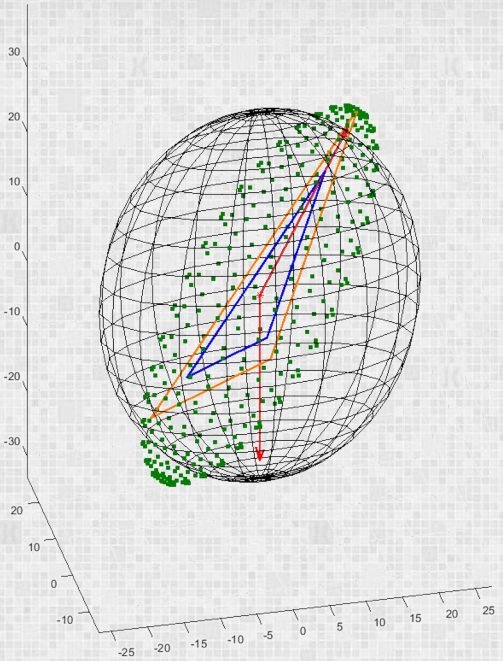


Cloud shape improvement
(Level of condensation)

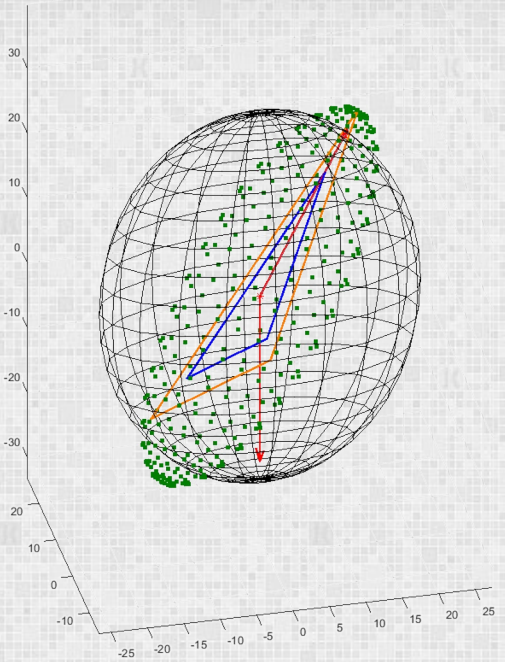


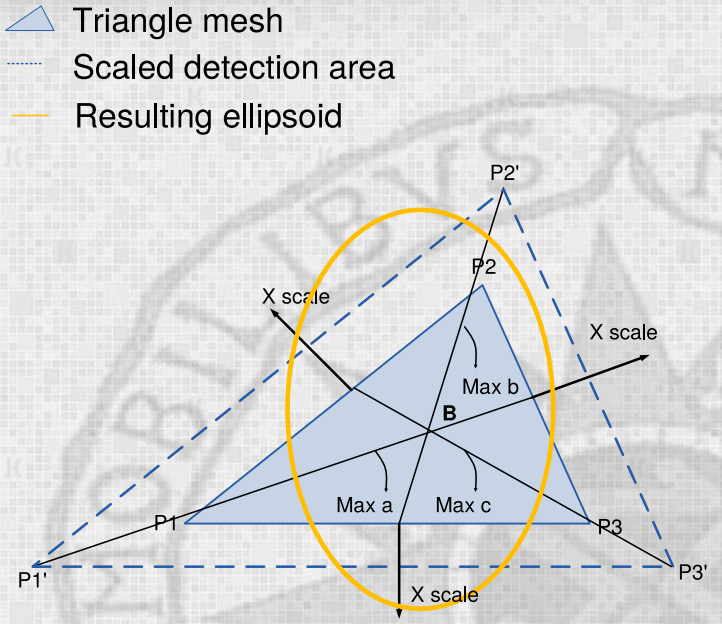
$$(!bFlat \vee (rayPos.y > gridMin.y + voxel[rayPos] \times \delta))) \implies render$$

Cloud shape improvement
(Clouds from 3D meshes)



Cloud shape improvement
(Clouds from 3D meshes)



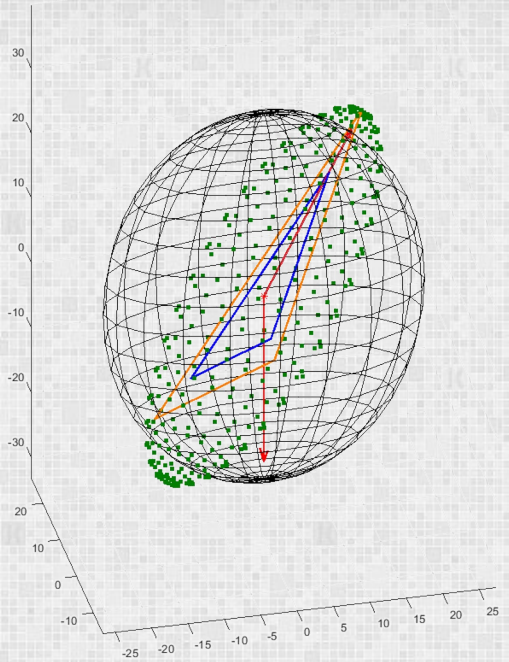


Cloud shape improvement
(Clouds from 3D meshes)

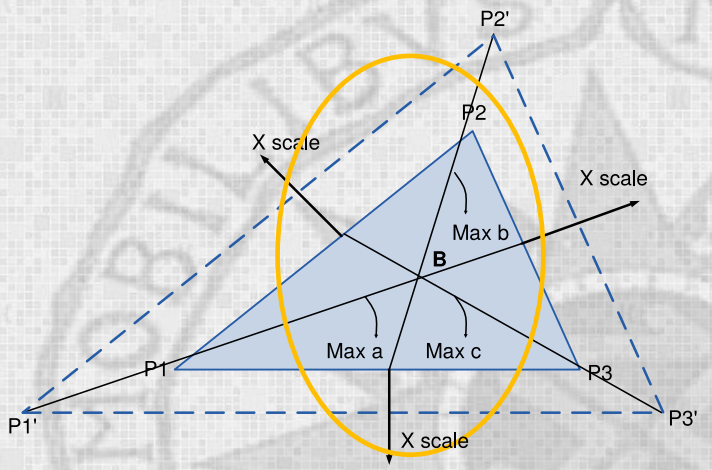
$$B = \left(\sum_{i=1}^{3} \frac{x_i}{3}, \sum_{i=1}^{3} \frac{y_i}{3}, \sum_{i=1}^{3} \frac{z_i}{3} \right) \quad \textbf{(2.5)}$$

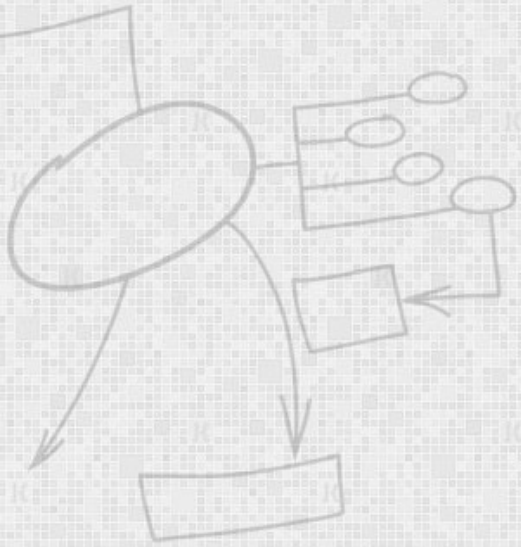
$$P'_i = (P_i - B) \cdot scale + B \quad \textbf{(2.6)}$$

$$radius_a = \|(B - P'_1)\|$$
$$radius_b = \|(B - P'_2)\| \quad \textbf{(2.7)}$$
$$radius_c = \|(B - P'_3)\|$$



Triangle mesh
Scaled detection area
Resulting ellipsoid



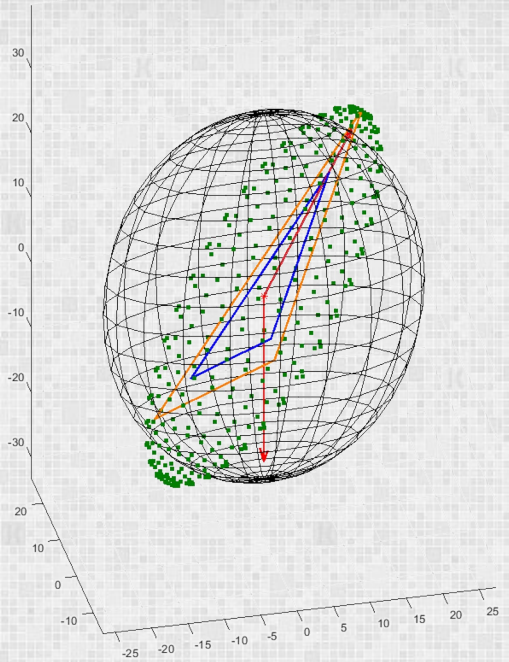


Cloud shape improvement
(Clouds from 3D meshes)

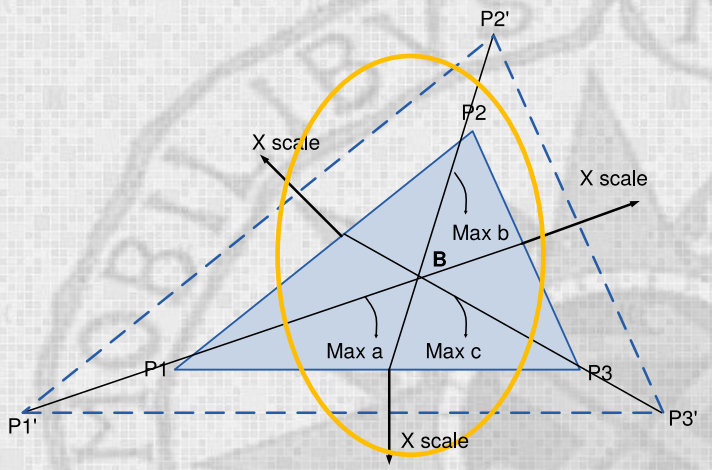
$$B = \left(\sum_{i=1}^{3}\frac{x_i}{3}, \sum_{i=1}^{3}\frac{y_i}{3}, \sum_{i=1}^{3}\frac{z_i}{3}\right) \quad (2.5)$$

$$P'_i = (P_i - B)\cdot scale + B \quad (2.6)$$

$$radius_a = \|(B - P'_1)\|$$
$$radius_b = \|(B - P'_2)\| \quad (2.7)$$
$$radius_c = \|(B - P'_3)\|$$



Triangle mesh
Scaled detection area
Resulting ellipsoid



$$if \begin{cases} max(radius_a) & dir\vec{T}ria = P'_1 - B \\ max(radius_b) & dir\vec{T}ria = P'_2 - B \\ max(radius_c) & dir\vec{T}ria = P'_3 - B \end{cases} \quad (2.8)$$

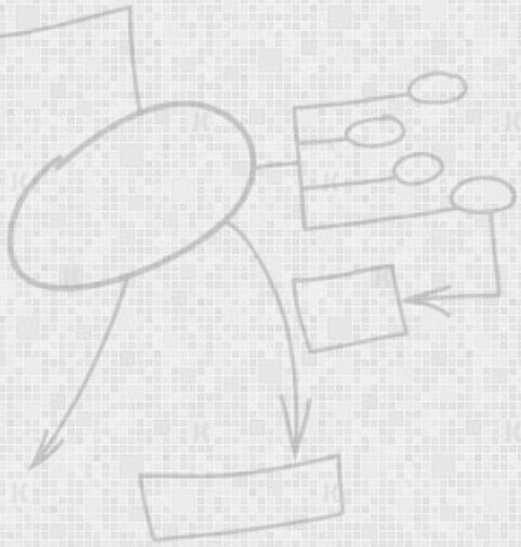
$$if \begin{cases} max(radius_a) & dir\vec{E}llip = (radius_a + B_x, B_y, B_z) - B \\ max(radius_b) & dir\vec{E}llip = (B_x, radius_b + B_y, B_z) - B \\ max(radius_c) & dir\vec{E}llip = (B_x, B_y, radius_c + B_z) - B \end{cases} \quad (2.9)$$

Rodrigues' Rotation Formula:

(1) Axis and angle using cross product and dot product:

$$x = \frac{dir\vec{E}llip \times dir\vec{T}ria}{\|dir\vec{E}llip \times dir\vec{T}ria\|} \qquad \textbf{(2.10)}$$

$$\theta = cos^{-1}\left(\frac{dir\vec{E}llip \cdot dir\vec{T}ria}{\|dir\vec{E}llip\| \cdot \|dir\vec{T}ria\|}\right)$$



Cloud shape improvement
(Clouds from 3D meshes)

Rodrigues' Rotation Formula:

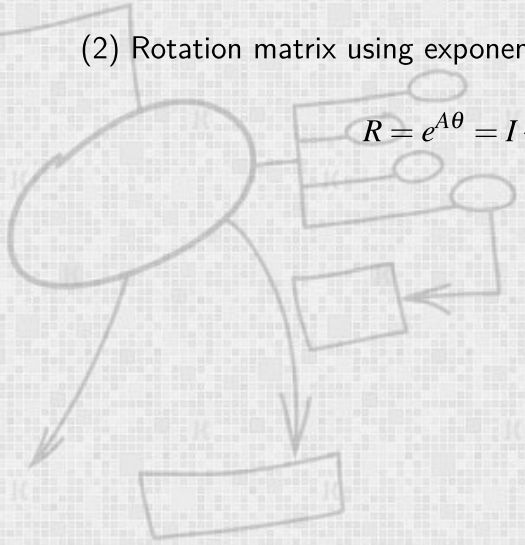
(1) Axis and angle using cross product and dot product:

$$x = \frac{dir\vec{E}llip \times dir\vec{T}ria}{\|dir\vec{E}llip \times dir\vec{T}ria\|} \quad \textbf{(2.10)}$$

$$\theta = cos^{-1}\left(\frac{dir\vec{E}llip \cdot dir\vec{T}ria}{\|dir\vec{E}llip\| \cdot \|dir\vec{T}ria\|}\right)$$

(2) Rotation matrix using exponential map:

$$R = e^{A\theta} = I + sin(\theta)\cdot A + (1 - cos(\theta))\cdot A^2 \quad \textbf{(2.11)}$$



Cloud shape improvement
(Clouds from 3D meshes)

Rodrigues' Rotation Formula:

(1) Axis and angle using cross product and dot product:

$$x = \frac{dir\vec{E}llip \times dir\vec{T}ria}{\|dir\vec{E}llip \times dir\vec{T}ria\|} \quad \textbf{(2.10)}$$

$$\theta = cos^{-1}\left(\frac{dir\vec{E}llip \cdot dir\vec{T}ria}{\|dir\vec{E}llip\| \cdot \|dir\vec{T}ria\|}\right)$$

(2) Rotation matrix using exponential map:

$$R = e^{A\theta} = I + sin(\theta)\cdot A + (1 - cos(\theta))\cdot A^2 \quad \textbf{(2.11)}$$

(3) A is a skew-symmetric matrix corresponding to x: **(2.12)**

$$A = [x]_x = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}$$

Cloud shape improvement
(Clouds from 3D meshes)

Rodrigues' Rotation Formula:

(1) Axis and angle using cross product and dot product:

$$x = \frac{dir\vec{E}llip \times dir\vec{T}ria}{\|dir\vec{E}llip \times dir\vec{T}ria\|} \quad \textbf{(2.10)}$$

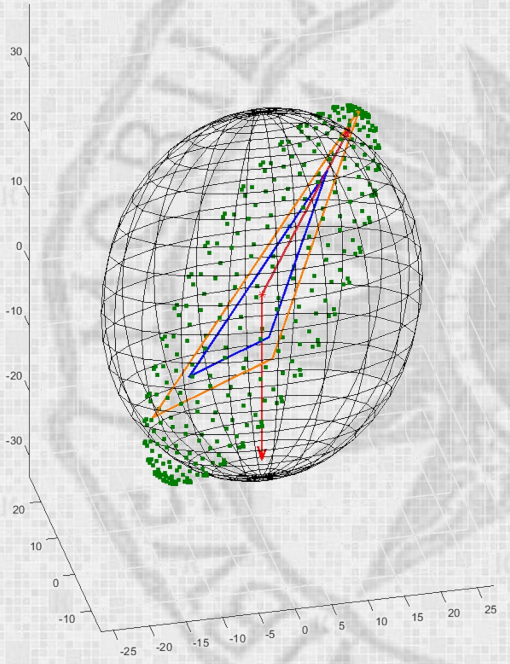
$$\theta = cos^{-1}\left(\frac{dir\vec{E}llip \cdot dir\vec{T}ria}{\|dir\vec{E}llip\| \cdot \|dir\vec{T}ria\|}\right)$$

(2) Rotation matrix using exponential map:

$$R = e^{A\theta} = I + sin(\theta)\cdot A + (1 - cos(\theta))\cdot A^2 \quad \textbf{(2.11)}$$

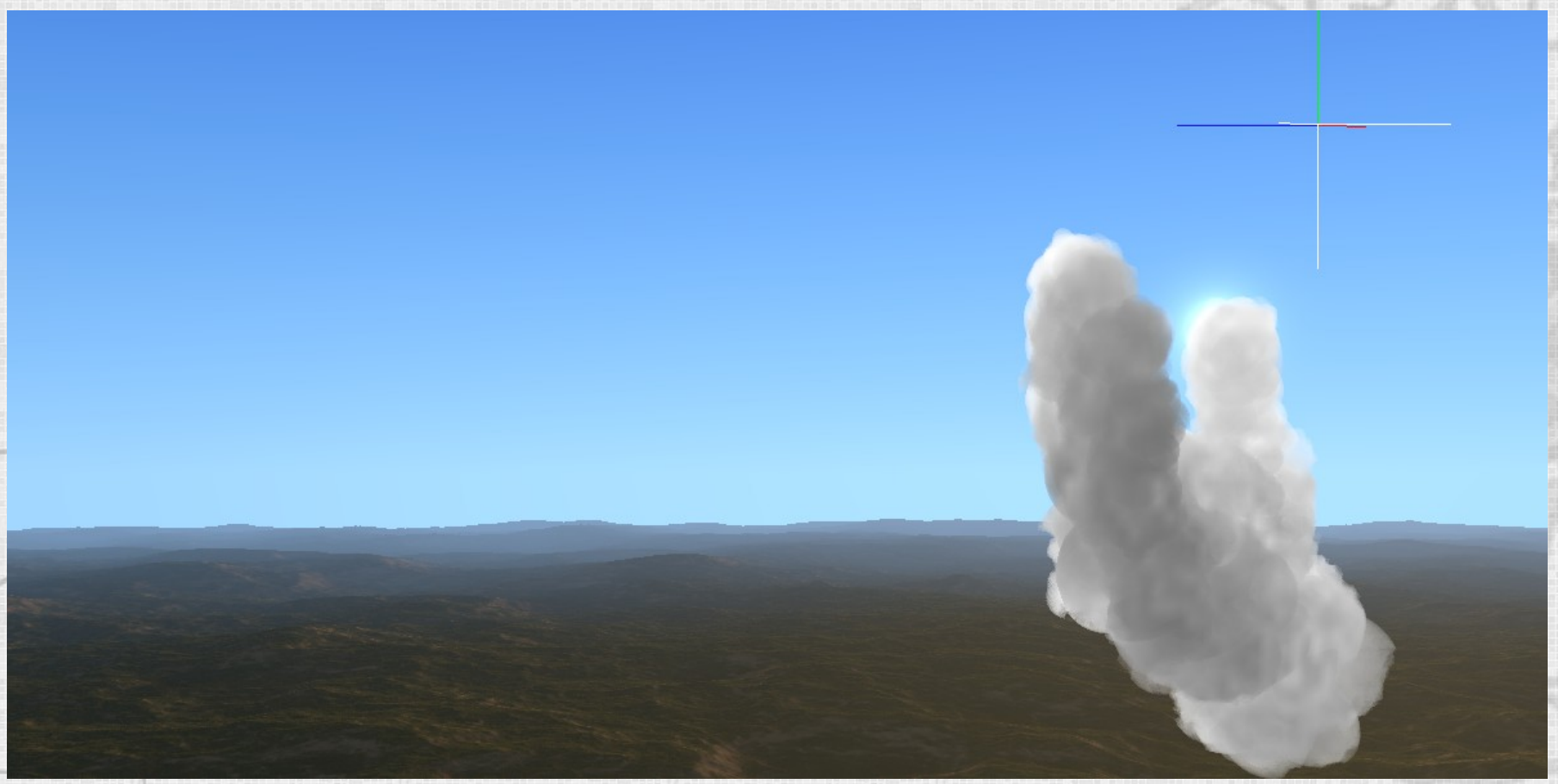
(3) A is a skew-symmetric matrix corresponding to x: **(2.12)**

$$A = [x]_x = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}$$



Cloud shape improvement
(Clouds from 3D meshes)

Rodrigues' Rotation Formula:



PART III

Fluid dynamics model

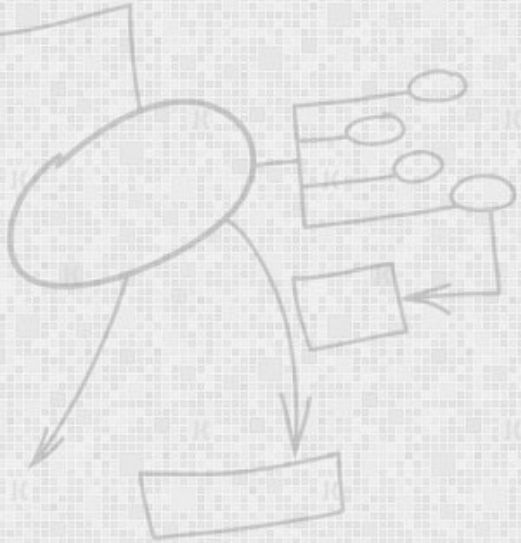
Based on Navier-Stokes equations:

$$\nabla v = 0 \tag{3.1}$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + v\nabla^2 u + F \tag{3.2}$$

$$\frac{\partial p}{\partial t} = -(u \cdot \nabla)\rho + k\nabla^2 \rho + S \tag{3.3}$$

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \tag{3.4}$$



Fluid dynamics model

Based on Navier-Stokes equations:

$$\nabla v = 0 \tag{3.1}$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + v\nabla^2 u + F \tag{3.2}$$

$$\frac{\partial p}{\partial t} = -(u \cdot \nabla)\rho + k\nabla^2 \rho + S \tag{3.3}$$

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \tag{3.4}$$

```
1 while simulating do
2 |
3 |   Retrieve UVW force equations
4 |   Add force
5 |   Diffuse
6 |   Project
7 |   Advect
8 |   Apply to guide points
9 end
```

Based on the 2D serial method of Jos Stam [Sta03]

Fluid dynamics model

Based on Navier-Stokes equations:

$$\nabla v = 0 \tag{3.1}$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + v\nabla^2 u + F \tag{3.2}$$

$$\frac{\partial p}{\partial t} = -(u \cdot \nabla)\rho + k\nabla^2 \rho + S \tag{3.3}$$

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \tag{3.4}$$

- **Add force:** $u = u_0 + \Delta t \times F.$ (3.5)

```
1 while simulating do
2 |
3 |   Retrieve UVW force equations
4 |   Add force
5 |   Diffuse
6 |   Project
7 |   Advect
8 |   Apply to guide points
9 end
```

Based on the 2D serial method of Jos Stam [Sta03]

Fluid dynamics model

Based on Navier-Stokes equations:

$$\nabla v = 0 \quad (3.1)$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + v\nabla^2 u + F \quad (3.2)$$

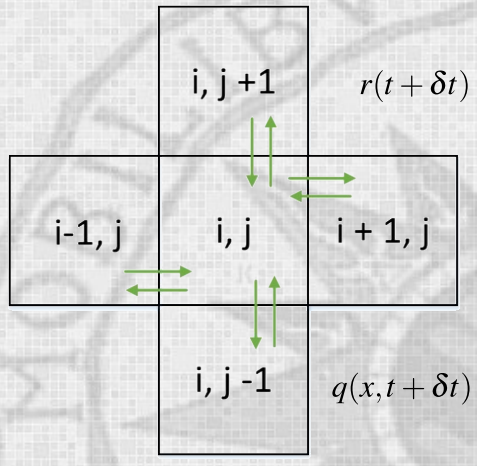
$$\frac{\partial p}{\partial t} = -(u \cdot \nabla)\rho + k\nabla^2 \rho + S \quad (3.3)$$

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \quad (3.4)$$

```
1 while simulating do
2 |
3 |   Retrieve UVW force equations
4 |   Add force
5 |   Diffuse
6 |   Project
7 |   Advect
8 |   Apply to guide points
9 end
```

Based on the 2D serial method of Jos Stam [Sta03]

- **Add force:** $u = u_0 + \Delta t \times F.$ (3.5)
- **Advection:**



$$r(t+\delta t) = r(t) + u(t)\delta t \quad (3.6)$$

$$q(x, t+\delta t) = q(x - u(x,t)\delta t, t) \quad (3.7)$$

Fluid dynamics model

Based on Navier-Stokes equations:

$$\nabla v = 0 \quad (3.1)$$

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + v\nabla^2 u + F \quad (3.2)$$

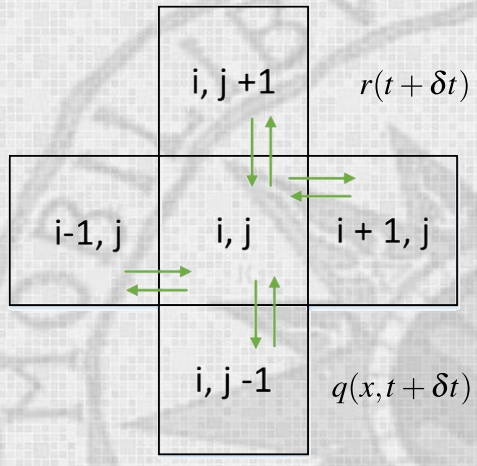
$$\frac{\partial p}{\partial t} = -(u \cdot \nabla)\rho + k\nabla^2 \rho + S \quad (3.3)$$

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \quad (3.4)$$

```
1 while simulating do
2 |
3 |   Retrieve UVW force equations
4 |   Add force
5 |   Diffuse
6 |   Project
7 |   Advect
8 |   Apply to guide points
9 end
```

Based on the 2D serial method of Jos Stam [Sta03]

- **Add force:** $u = u_0 + \Delta t \times F.$ (3.5)
- **Advection:**



$$r(t+\delta t) = r(t) + u(t)\delta t \quad (3.6)$$

$$q(x, t+\delta t) = q(x - u(x,t)\delta t, t) \quad (3.7)$$

- **Difussion:**

$$u(x, t+\delta t) = u(x,t) + v\delta t \nabla^2 u(x,t) \quad (3.8)$$

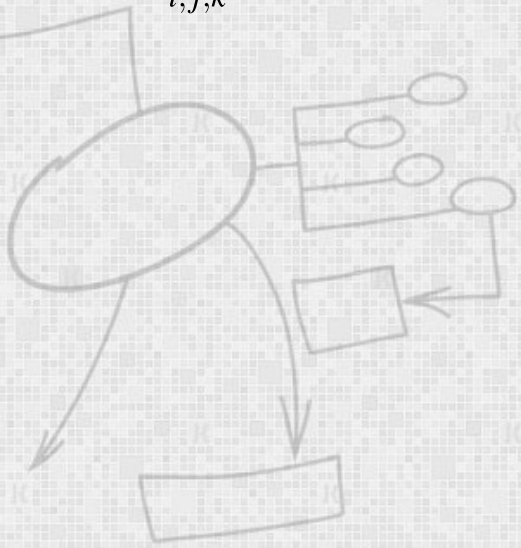
$$(I - v\delta t\nabla^2)u(x, t+\delta t) = u(x,t) \quad (3.9)$$

Fluid class parallelization

$$Block_{x,y,z} = \left\lceil \frac{Dim_{x,y,z} + 2}{T_{x,y,z}} \right\rceil \tag{3.10}$$

$$D^{n}_{i,j,k} = D^{n+1}_{i,j,k} - \frac{kdt}{h^3}\left(D^{n+1}_{i-1,j,k} + D^{n+1}_{i,j-1,k} + D^{n+1}_{i,j,k-1}, D^{n+1}_{i+1,j,k} + D^{n+1}_{i,j+1,k} + D^{n+1}_{i,j,k+1} - 6D^{n+1}_{i,j,k}\right) \tag{3.11}$$

$$D^{n+1}_{i,j,k} = \frac{D^{n}_{i,j,k} + \frac{kdt}{h^3}\left(D^{n+1}_{i-1,j,k} + D^{n+1}_{i,j-1,k} + D^{n+1}_{i,j,k-1}, D^{n+1}_{i+1,j,k} + D^{n+1}_{i,j+1,k} + D^{n+1}_{i,j,k+1} + D^{n+1}_{i,j,k}\right)}{1 + \frac{kdt}{h^3}} \tag{3.12}$$



Fluid class parallelization

$$Block_{x,y,z} = \left\lceil \frac{Dim_{x,y,z}+2}{T_{x,y,z}} \right\rceil \tag{3.10}$$

$$D^{n}_{i,j,k} = D^{n+1}_{i,j,k} - \frac{kdt}{h^3}\left(D^{n+1}_{i-1,j,k} + D^{n+1}_{i,j-1,k} + D^{n+1}_{i,j,k-1}, D^{n+1}_{i+1,j,k} + D^{n+1}_{i,j+1,k} + D^{n+1}_{i,j,k+1} - 6D^{n+1}_{i,j,k}\right) \tag{3.11}$$

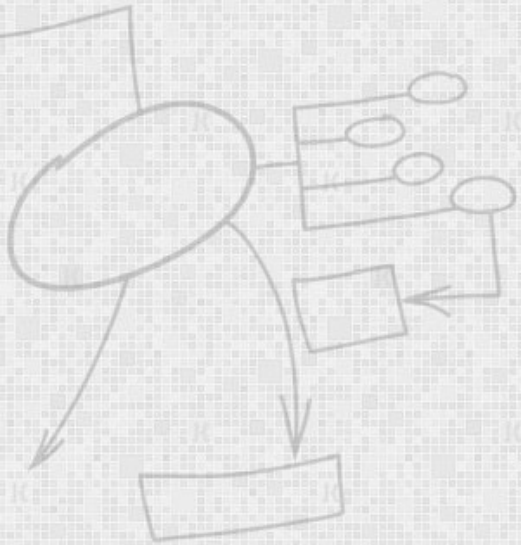
$$D^{n+1}_{i,j,k} = \frac{D^{n}_{i,j,k} + \frac{kdt}{h^3}\left(D^{n+1}_{i-1,j,k} + D^{n+1}_{i,j-1,k} + D^{n+1}_{i,j,k-1}, D^{n+1}_{i+1,j,k} + D^{n+1}_{i,j+1,k} + D^{n+1}_{i,j,k+1} + D^{n+1}_{i,j,k}\right)}{1+\frac{kdt}{h^3}} \tag{3.12}$$

```
1  Function linearSolverKernel(x,x0,y,a,c)
2  |  i,j,k ← blockIdx_{x,y,z} × blockDim_{x,y,z} + threadIdx_{x,y,z}
3  |  if ((i,j,k ≥ 1) and (i,j,k ≤ Dim_{x,y,z})) then
4  |  |  y[i,j,k] ← x0[i,j,k] + a × (x[i−1,j,k] + x[i+1,j,k] + x[i,j−1,k] + x[i,j+1,k] +
   |  |    x[i,j,k−1] + x[i,j,k+1])/c
5  |  end
6
7  (...)
8
9  for t ← 1 < times for approximation do
10 |  linearSolverKernel <<< grid,block >>> (x,x0,y,a,x)
11 |  aux ← x
12 |  x ← y
13 |  y ← aux
14 end
```

Jacobi relaxation for the implicit linear solver

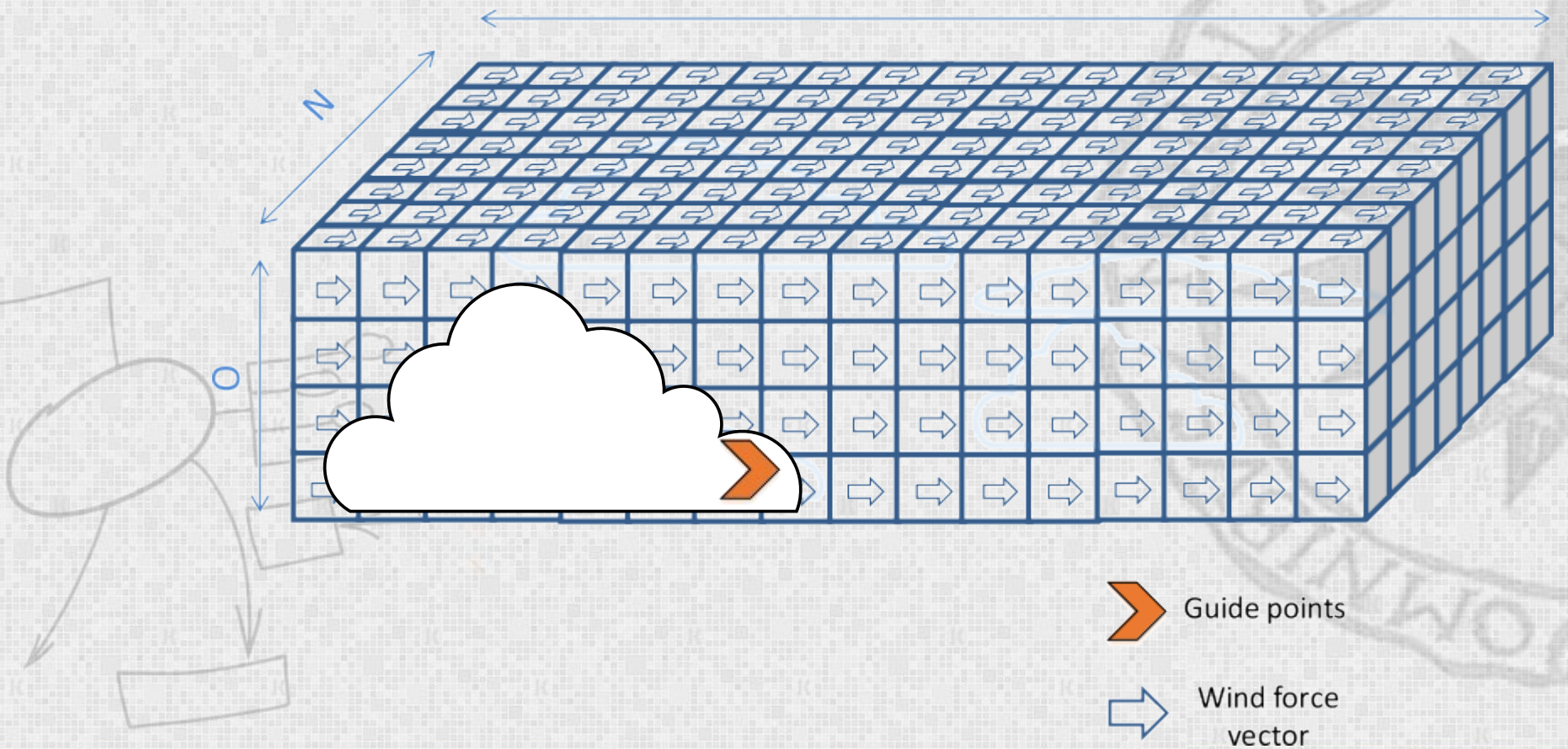
Guide points

$$sphPos(x,y,z)_i = \sum_{i=1}^{S} F_{U,V,W}(x,y,z)$$



Guide points

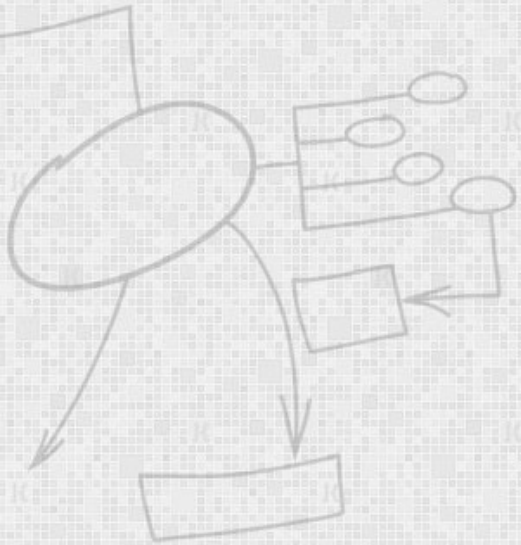
$$sphPos(x,y,z)_i = \sum_{i=1}^{S} F_{U,V,W}(x,y,z)$$



Cloud morphing model

Linear interpolation is required:

$$f(\vec{x}, \vec{y}, a) = \vec{x} \cdot (1 - a) + \vec{y} \cdot a$$

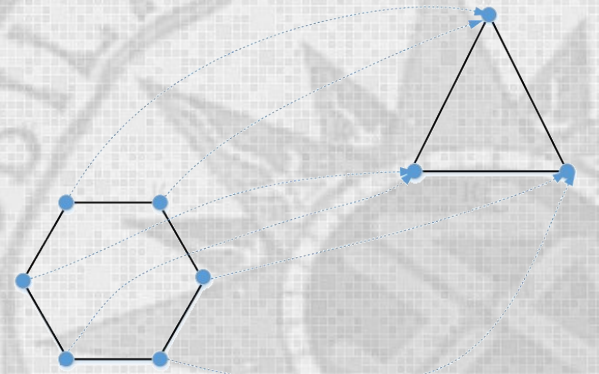


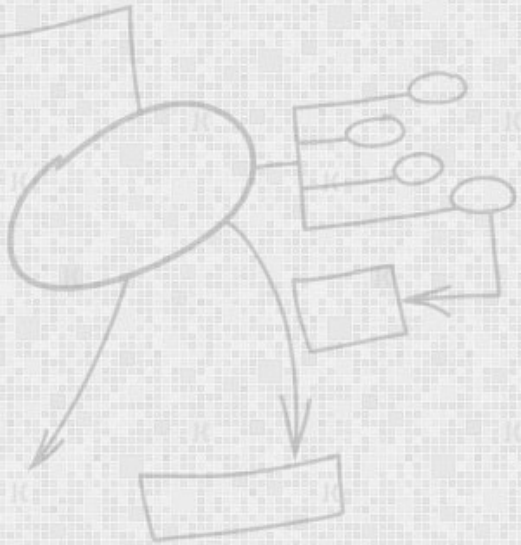
Cloud morphing model

Linear interpolation is required:

$$f(\vec{x}, \vec{y}, a) = \vec{x} \cdot (1 - a) + \vec{y} \cdot a$$

A) Barycenters in the source > Barycenters in the target



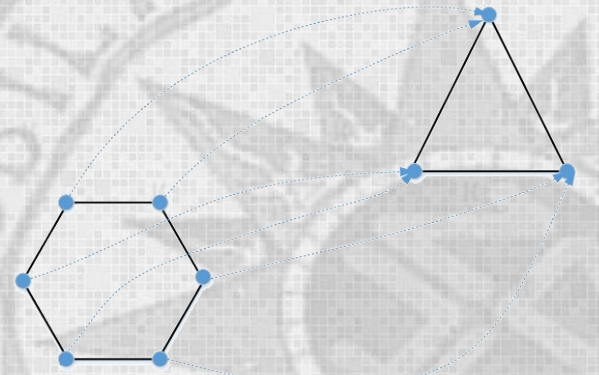


Cloud morphing model

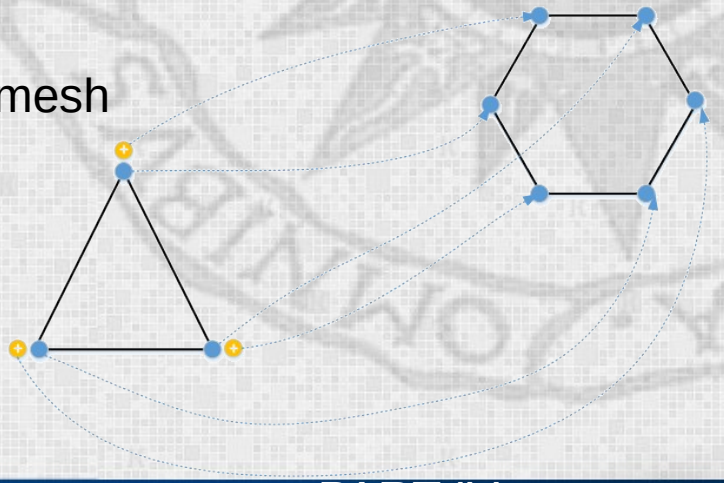
Linear interpolation is required:

$$f(\vec{x}, \vec{y}, a) = \vec{x} \cdot (1 - a) + \vec{y} \cdot a$$

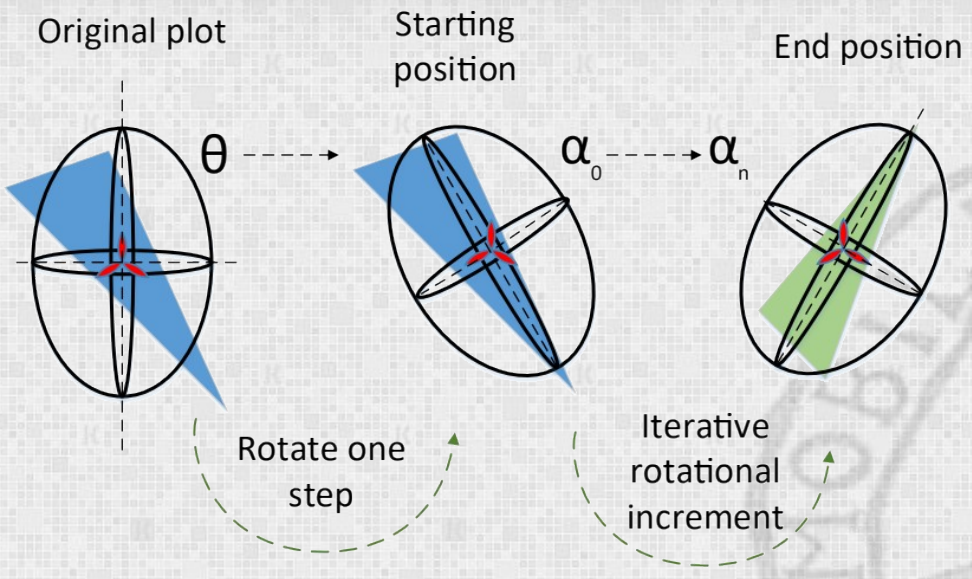
A) Barycenters in the source > Barycenters in the target



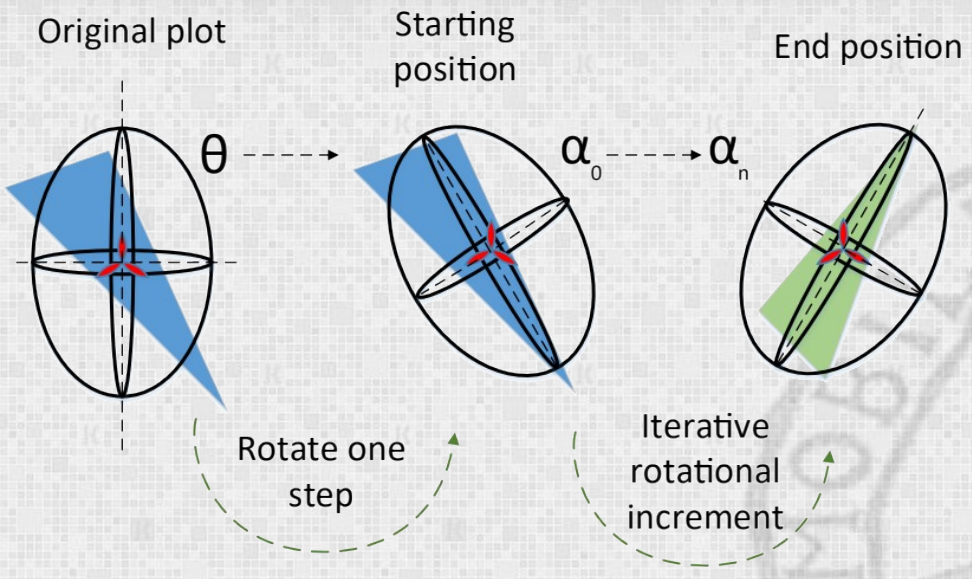
B) Barycenters in the target > Barycenters in the source mesh



Cloud morphing model



Cloud morphing model

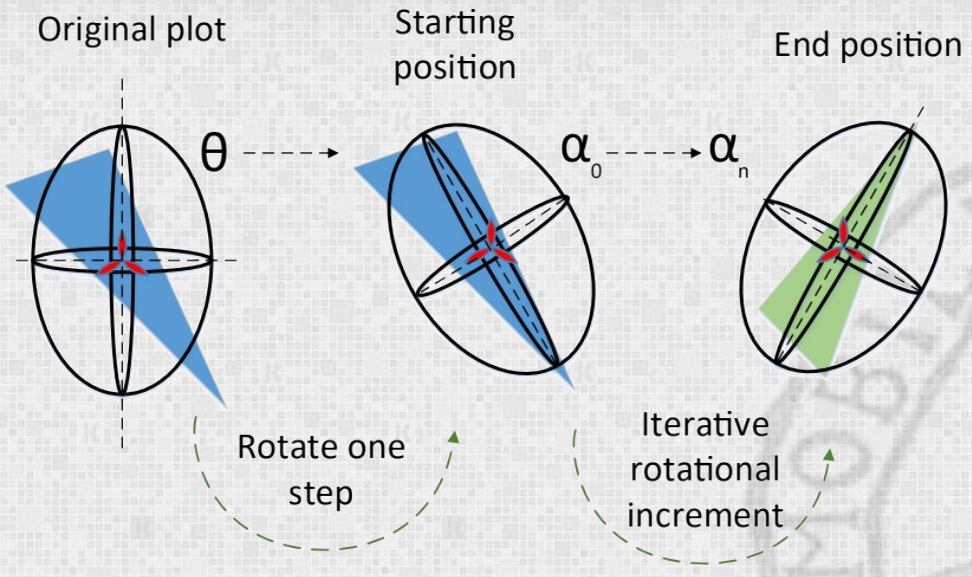


Rodrigues again:

$$R_1 = e^{A\theta} = I + sin(\theta)A + (1 - cos(\theta))A^2$$

$$R_2 = e^{A\alpha_i} = I + sin(\alpha_i)A + (1 - cos(\alpha_i))A^2$$

Cloud morphing model



Rodrigues again:

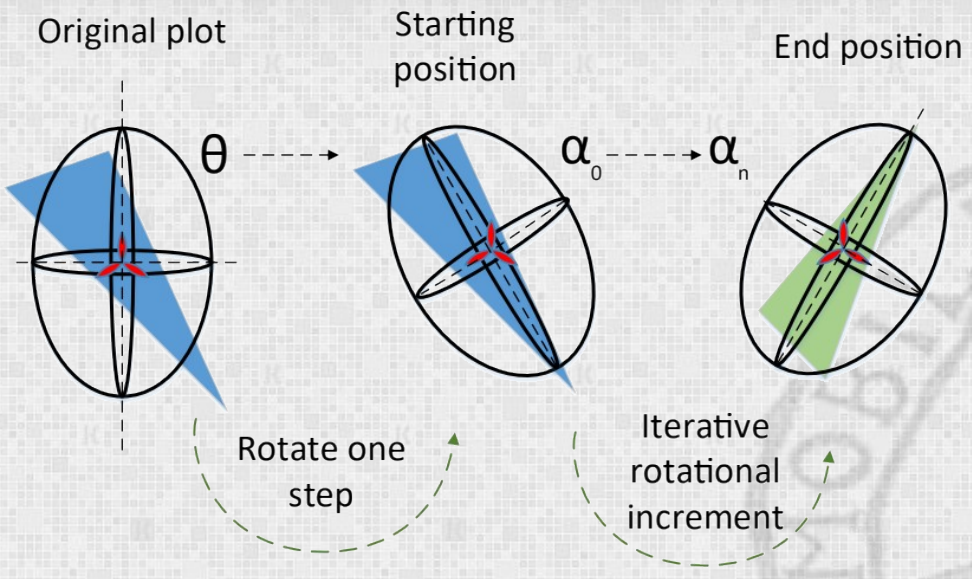
$$R_1 = e^{A\theta} = I + sin(\theta)A + (1 - cos(\theta))A^2$$

$$R_2 = e^{A\alpha_i} = I + sin(\alpha_i)A + (1 - cos(\alpha_i))A^2$$

Final transformation:

$$T_{final} = R_2 \cdot R_1 \cdot \text{Point of pseudoellipsoid}$$

Cloud morphing model



Rodrigues again:

$$R_1 = e^{A\theta} = I + sin(\theta)A + (1 - cos(\theta))A^2$$

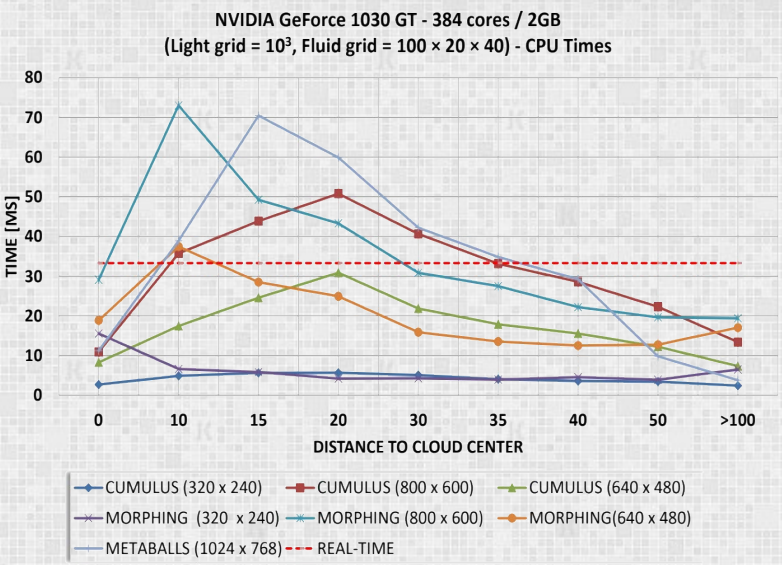
$$R_2 = e^{A\alpha_i} = I + sin(\alpha_i)A + (1 - cos(\alpha_i))A^2$$

Final transformation:

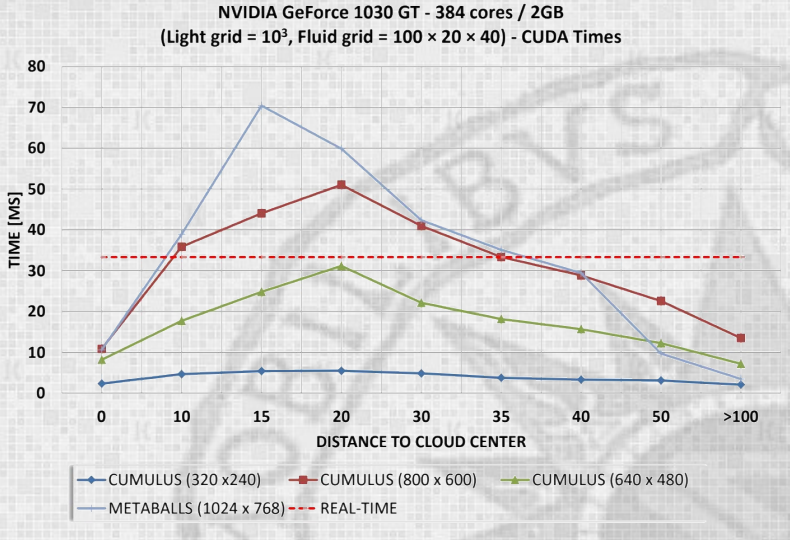
$$T_{final} = R_2 \cdot R_1 \cdot \text{Point of pseudoellipsoid}$$

PART IV

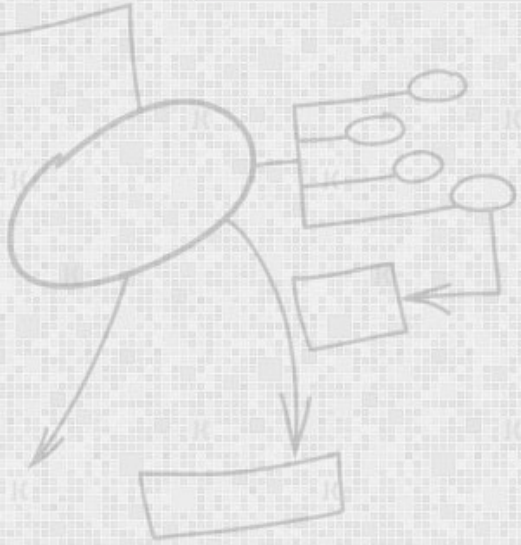
GPU benchmarks
(nVidia GeForce 1030GT)



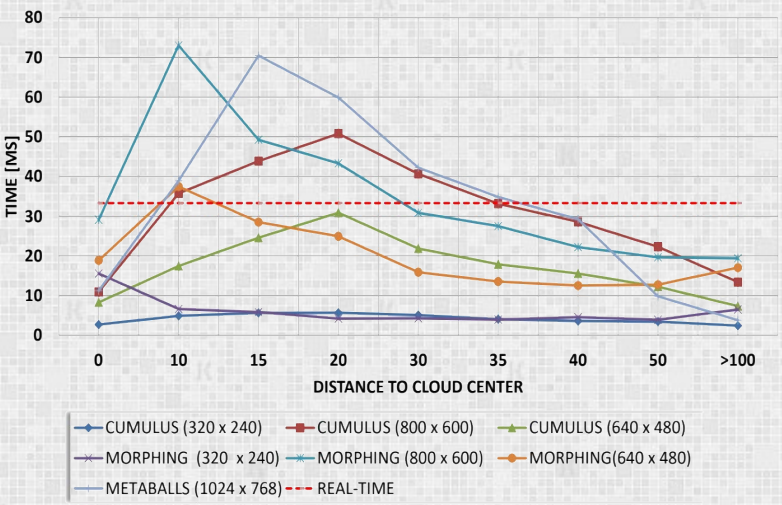
79% > 30 FPS



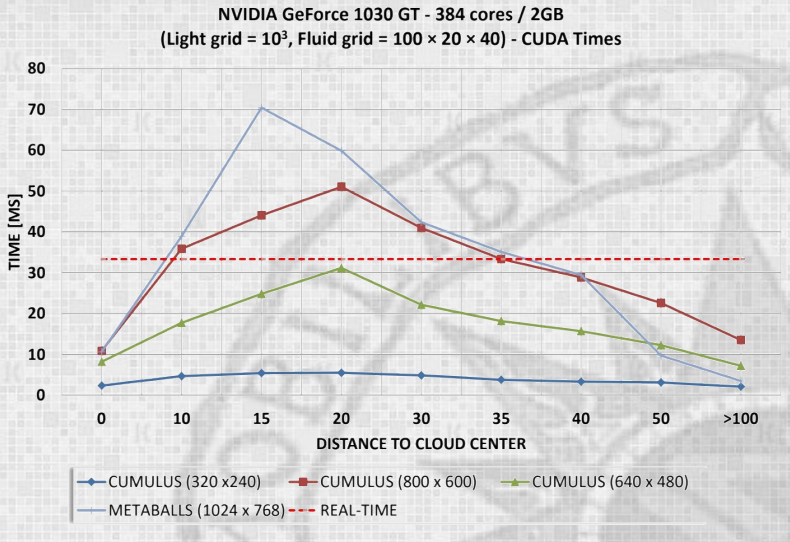
75% > 30 FPS



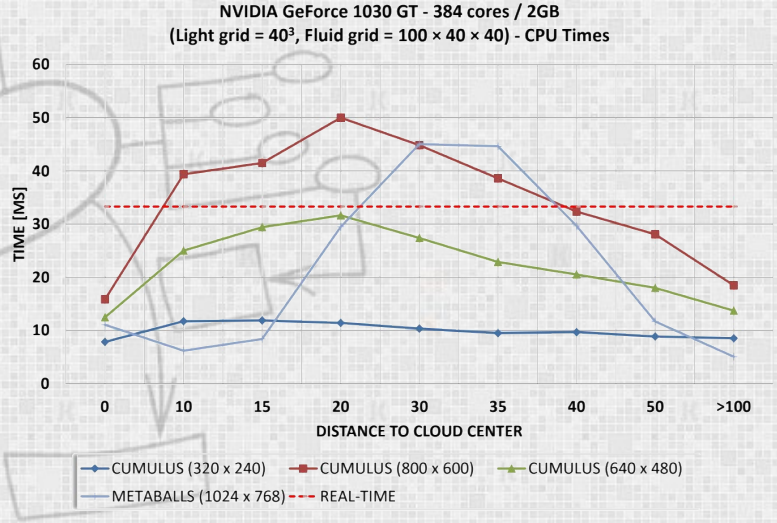
GPU benchmarks
(nVidia GeForce 1030GT)



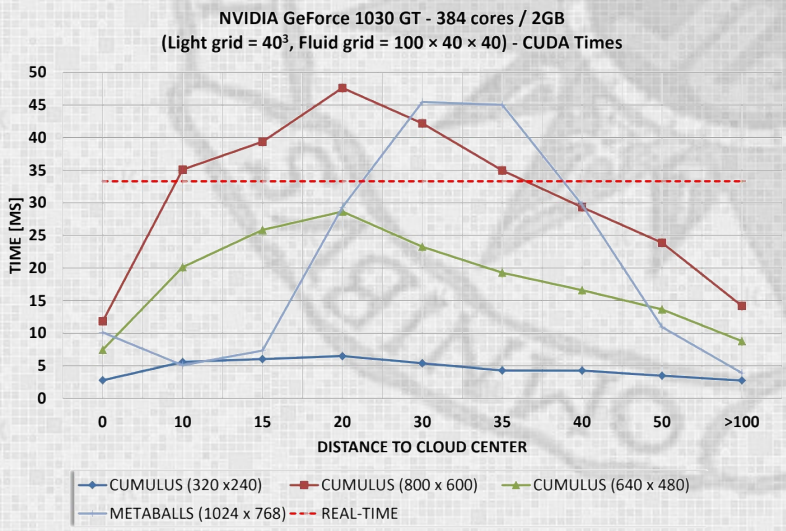
79% > 30 FPS



75% > 30 FPS

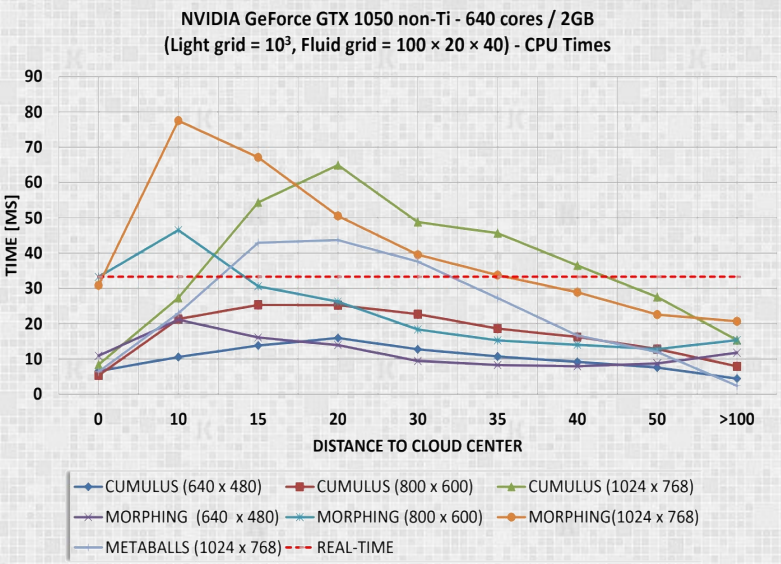


80% > 30 FPS

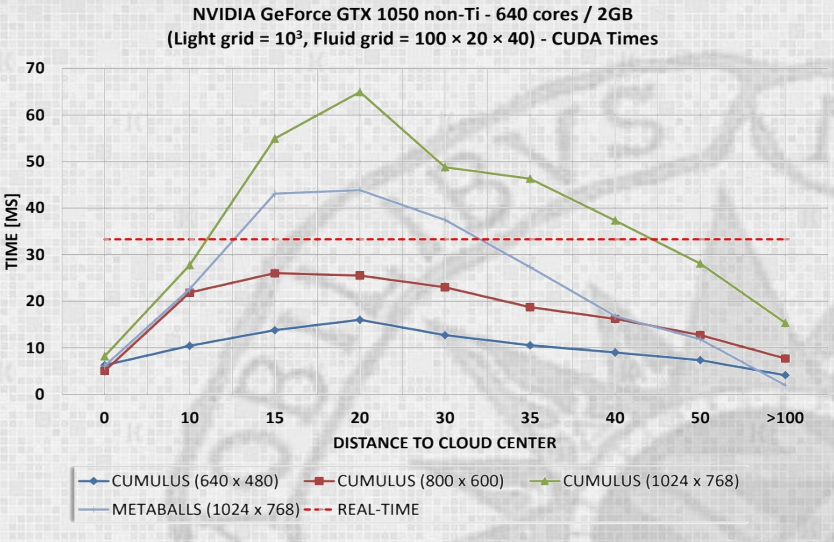


80% > 30 FPS

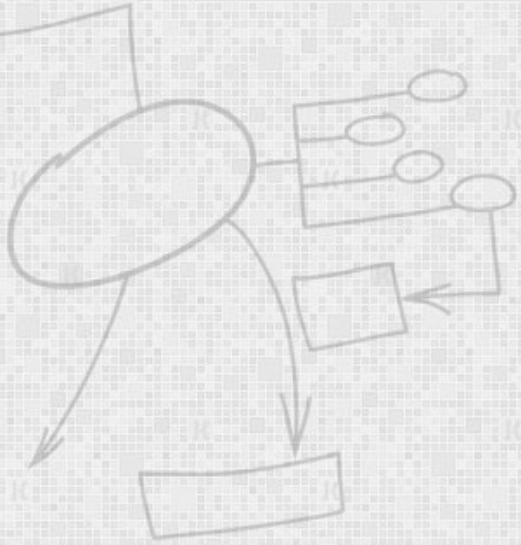
GPU benchmarks
(nVidia GeForce GTX1050 non-Ti)



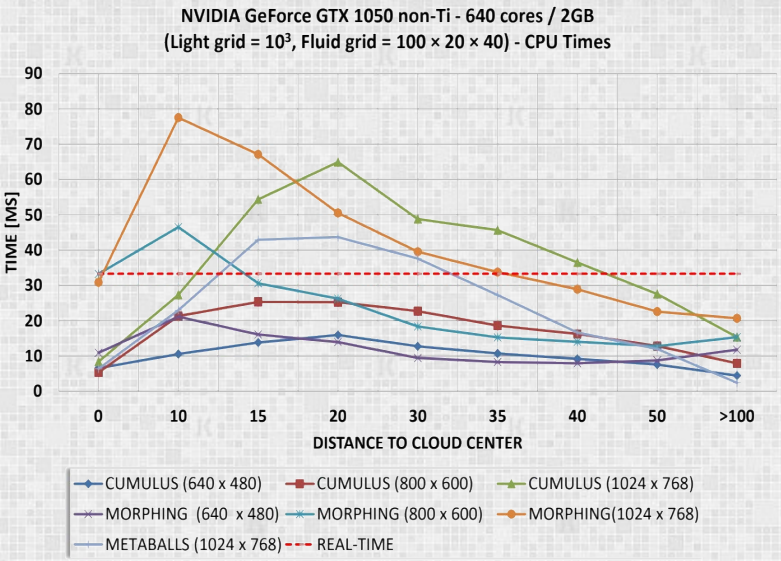
78% > 30 FPS



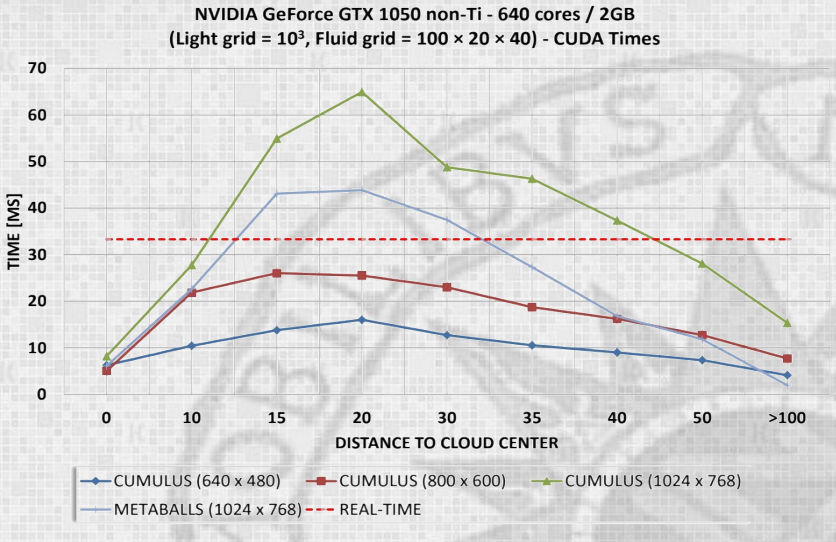
78% > 30 FPS



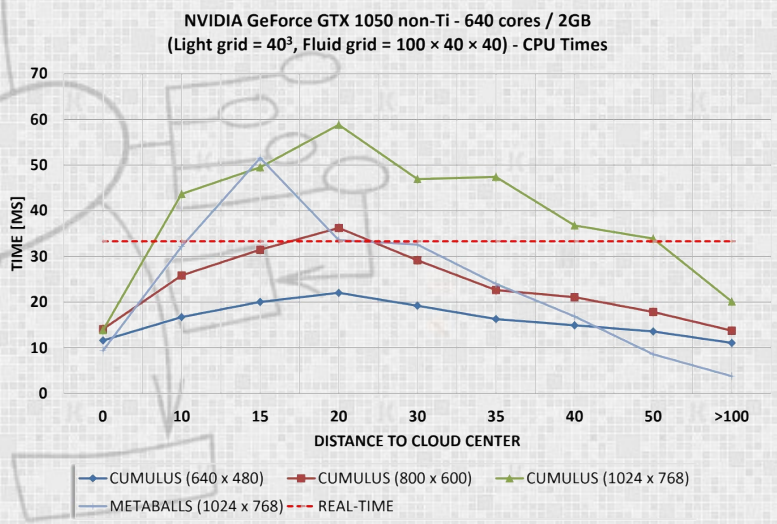
GPU benchmarks
(nVidia GeForce GTX1050 non-Ti)



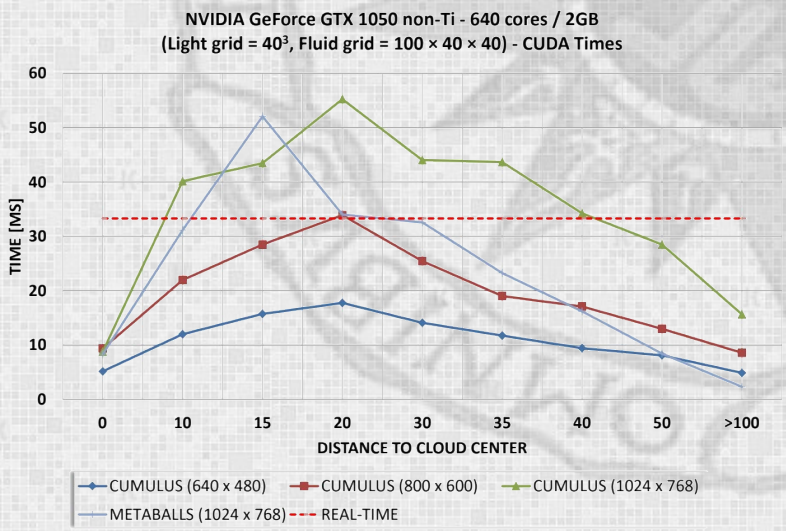
78% > 30 FPS



78% > 30 FPS

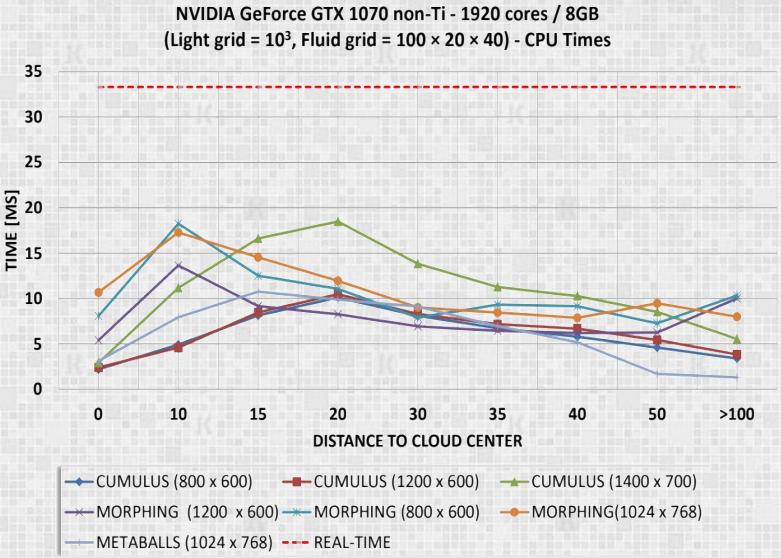


72% > 30 FPS

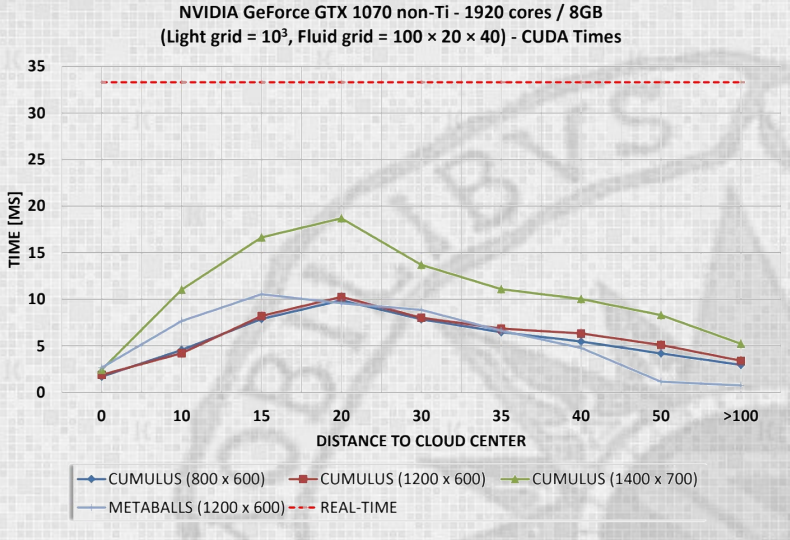


75% > 30 FPS

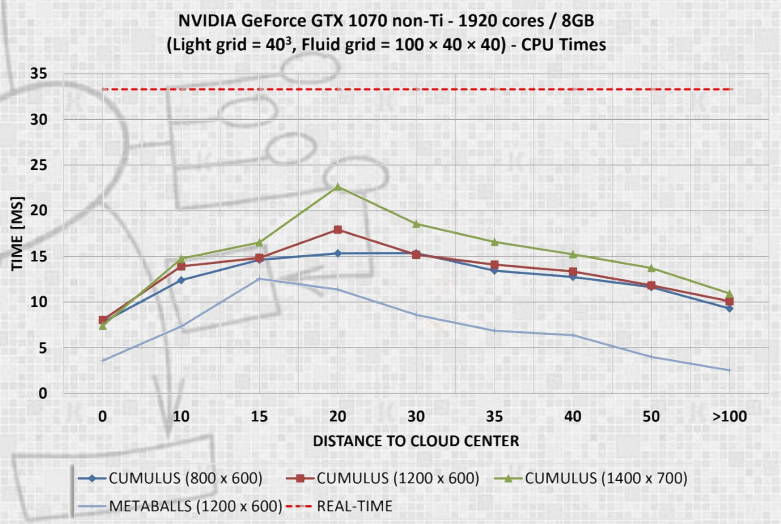
GPU benchmarks
(nVidia GeForce GTX1070 non-Ti)



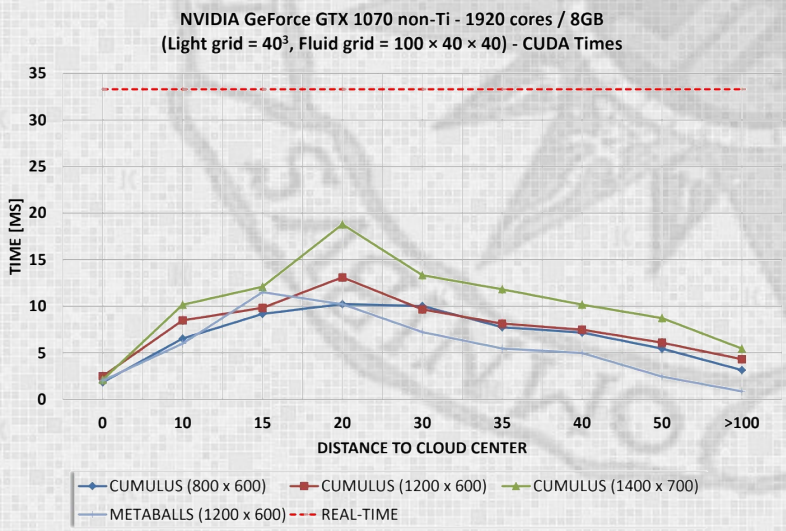
100% > 30 FPS



100% > 30 FPS

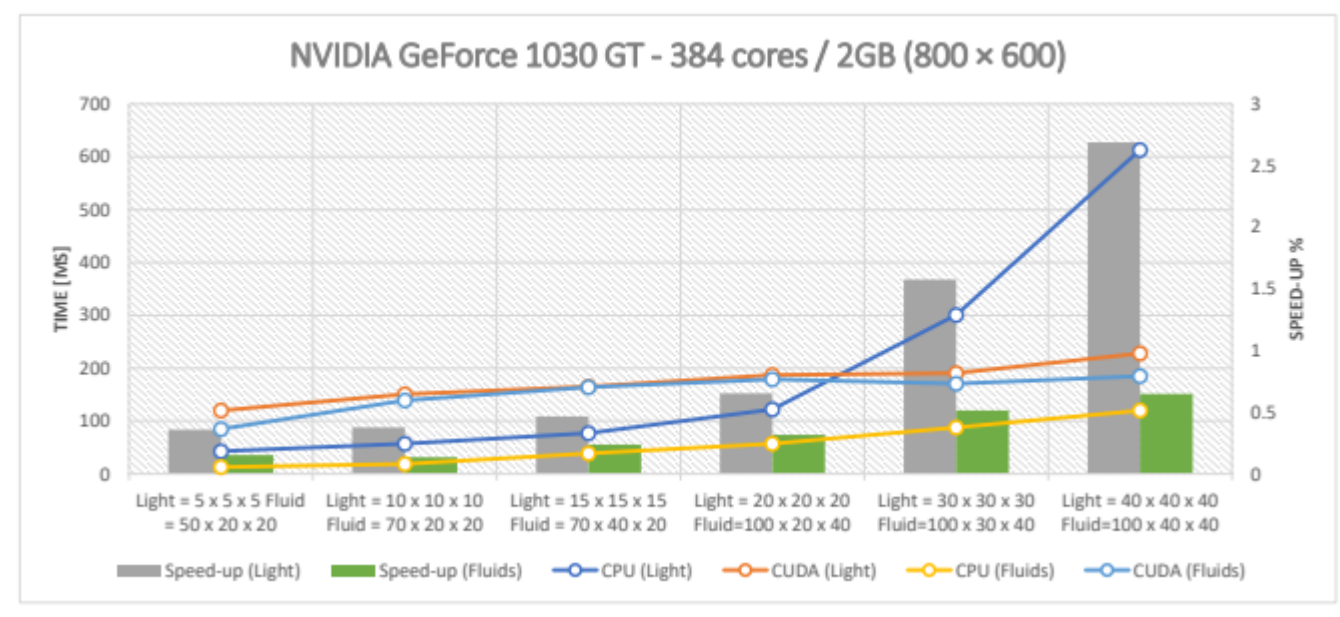


100% > 30 FPS

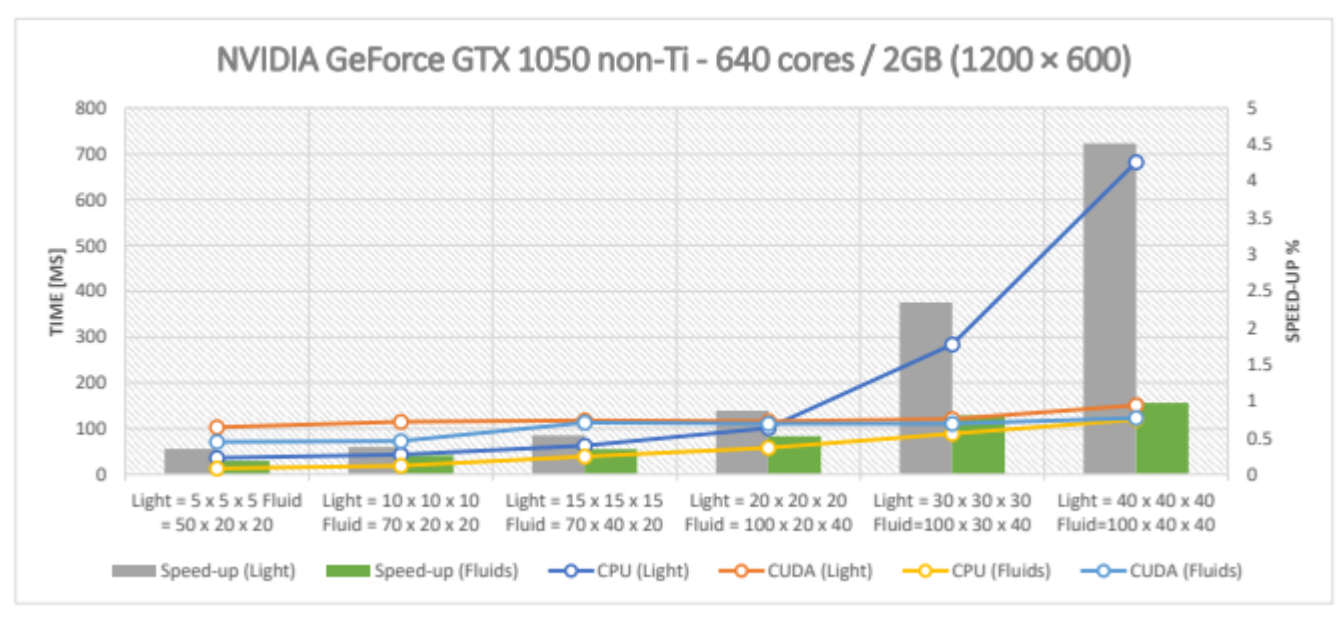


100% > 30 FPS

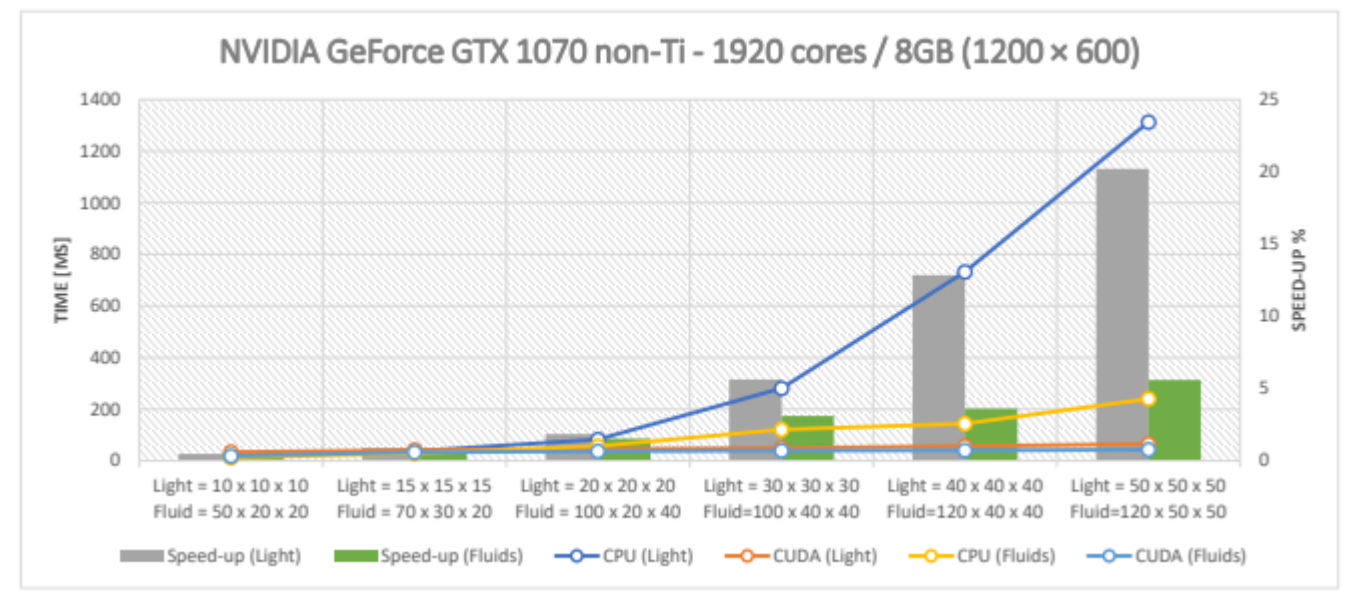
GPU benchmarks
(Speedup)



GPU benchmarks
(Speedup)

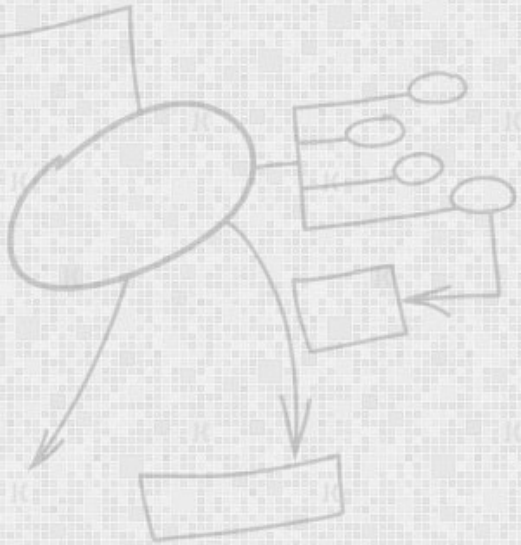


GPU benchmarks
(Speedup)



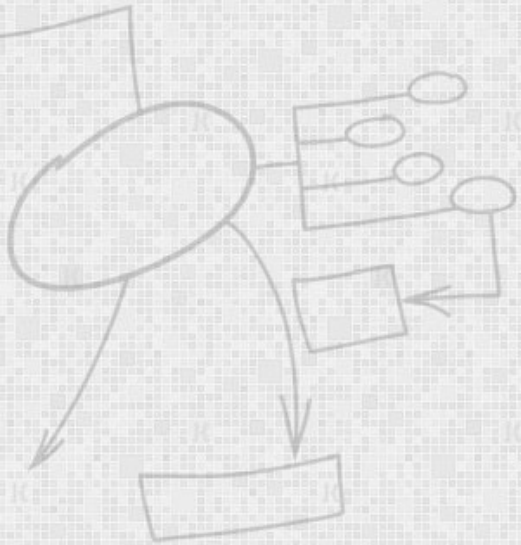
Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]



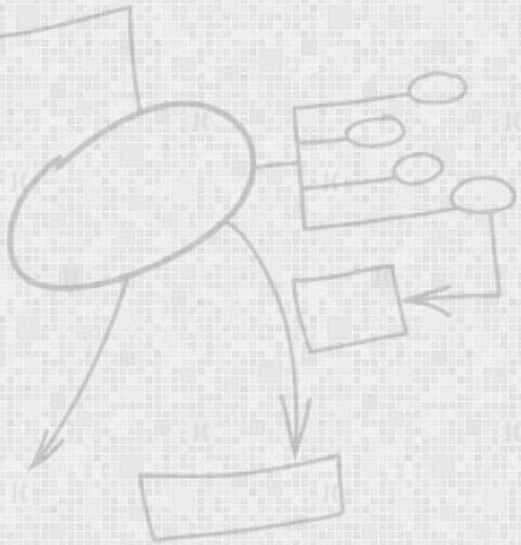
Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]
- More performance than photo-realistic methods by *Narasimhan et al.* [Nar+06], *Jarosz et al.* [Jar+08] and *Cerezo et al.* [Cer+05]



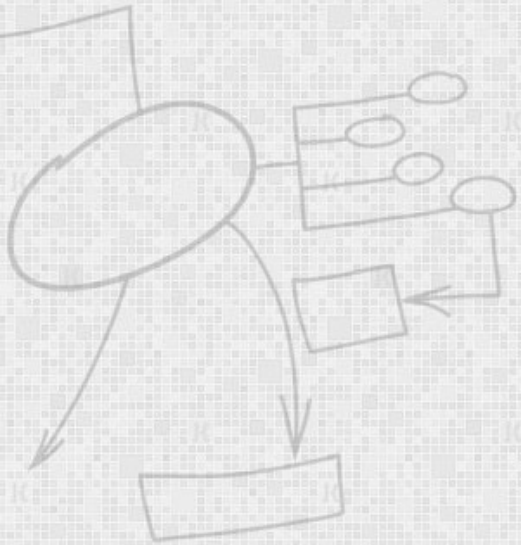
Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]
- More performance than photo-realistic methods by *Narasimhan et al.* [Nar+06], *Jarosz et al.* [Jar+08] and *Cerezo et al.* [Cer+05]
- Lower GPU overhead than the works of *Zhou et al.* [Zho+08]



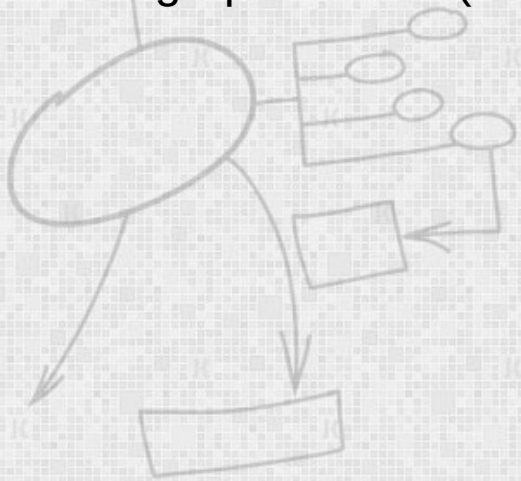
Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]
- More performance than photo-realistic methods by *Narasimhan et al.* [Nar+06], *Jarosz et al.* [Jar+08] and *Cerezo et al.* [Cer+05]
- Lower GPU overhead than the works of *Zhou et al.* [Zho+08]
- Better pre-computation time to the milliseconds level as opposed to minutes in *Yusov* [Yus14], *Zhou et al.* [Zho+08], *Ament, Sadlo and Weiskopf* [ASW13] and *Kallweit et al.* [Kal+17] (hours)



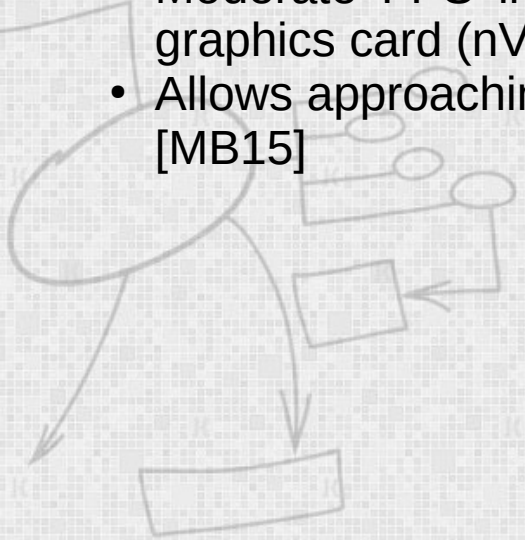
Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]
- More performance than photo-realistic methods by *Narasimhan et al.* [Nar+06], *Jarosz et al.* [Jar+08] and *Cerezo et al.* [Cer+05]
- Lower GPU overhead than the works of *Zhou et al.* [Zho+08]
- Better pre-computation time to the milliseconds level as opposed to minutes in *Yusov* [Yus14], *Zhou et al.* [Zho+08], *Ament, Sadlo and Weiskopf* [ASW13] and *Kallweit et al.* [Kal+17] (hours)
- Moderate FPS increase with respect the work of *Bouthors* [Bou+08] with the same graphics card (nVidia GeForce 8800 GTS – 96 cores) without the same hyper-realism



Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]
- More performance than photo-realistic methods by *Narasimhan et al.* [Nar+06], *Jarosz et al.* [Jar+08] and *Cerezo et al.* [Cer+05]
- Lower GPU overhead than the works of *Zhou et al.* [Zho+08]
- Better pre-computation time to the milliseconds level as opposed to minutes in *Yusov* [Yus14], *Zhou et al.* [Zho+08], *Ament, Sadlo and Weiskopf* [ASW13] and *Kallweit et al.* [Kal+17] (hours)
- Moderate FPS increase with respect the work of *Bouthors* [Bou+08] with the same graphics card (nVidia GeForce 8800 GTS – 96 cores) without the same hyper-realism
- Allows approaching, traversing, turning around in opposition to *Mukina* and Bezgodov [MB15]



Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]
- More performance than photo-realistic methods by *Narasimhan et al.* [Nar+06], *Jarosz et al.* [Jar+08] and *Cerezo et al.* [Cer+05]
- Lower GPU overhead than the works of *Zhou et al.* [Zho+08]
- Better pre-computation time to the milliseconds level as opposed to minutes in *Yusov* [Yus14], *Zhou et al.* [Zho+08], *Ament, Sadlo and Weiskopf* [ASW13] and *Kallweit et al.* [Kal+17] (hours)
- Moderate FPS increase with respect the work of *Bouthors* [Bou+08] with the same graphics card (nVidia GeForce 8800 GTS – 96 cores) without the same hyper-realism
- Allows approaching, traversing, turning around in opposition to *Mukina* and Bezgodov [MB15]
- Better 3D mesh smoothness and accuracy than the work of *Wither, Bouthors, and Can* [WBC08]

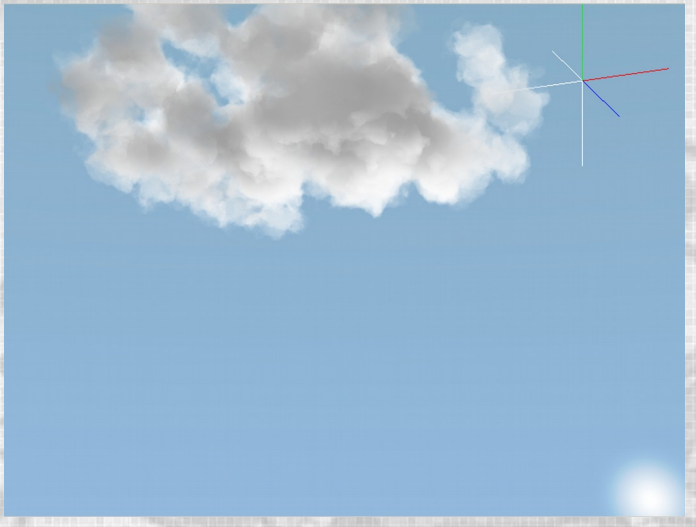
Discussions

- Greater realism than particle systems found in *Kang et al.* [KPK15], *Huang et al.* [Hua+08], *Harris* [Har02], *Bi et al.* [Bi+16], *Horng-Shyang et al.* [Hor+05] and *Montenegro et al.* [Mon+17]
- More performance than photo-realistic methods by *Narasimhan et al.* [Nar+06], *Jarosz et al.* [Jar+08] and *Cerezo et al.* [Cer+05]
- Lower GPU overhead than the works of *Zhou et al.* [Zho+08]
- Better pre-computation time to the milliseconds level as opposed to minutes in *Yusov* [Yus14], *Zhou et al.* [Zho+08], *Ament, Sadlo and Weiskopf* [ASW13] and *Kallweit et al.* [Kal+17] (hours)
- Moderate FPS increase with respect the work of *Bouthors* [Bou+08] with the same graphics card (nVidia GeForce 8800 GTS – 96 cores) without the same hyper-realism
- Allows approaching, traversing, turning around in opposition to *Mukina* and Bezgodov [MB15]
- Better 3D mesh smoothness and accuracy than the work of *Wither, Bouthors, and Can* [WBC08]
- Morphing effect with better performance than the model proposed by *Yu* and *Wang* [YW11] (45-95 vs. 80-137 FPS with similar graphic cards)

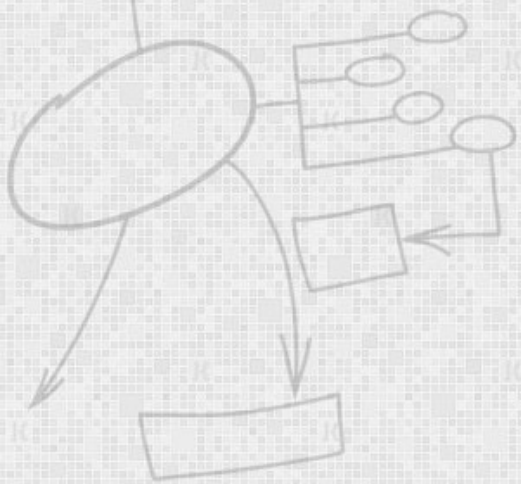
Discussions

Harris [Har02], *Huang et al.* [Hua+08]

Montenegro et al. [Mon+17]



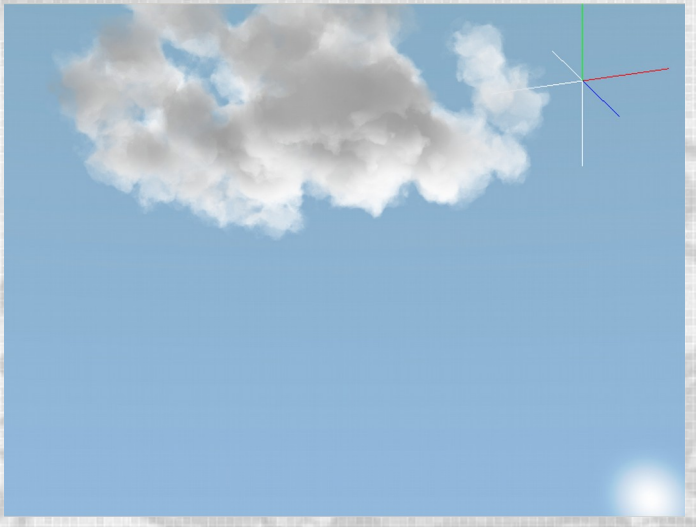
Proposed method



Discussions

Harris [Har02], *Huang et al.* [Hua+08]

Montenegro et al. [Mon+17]



Proposed method

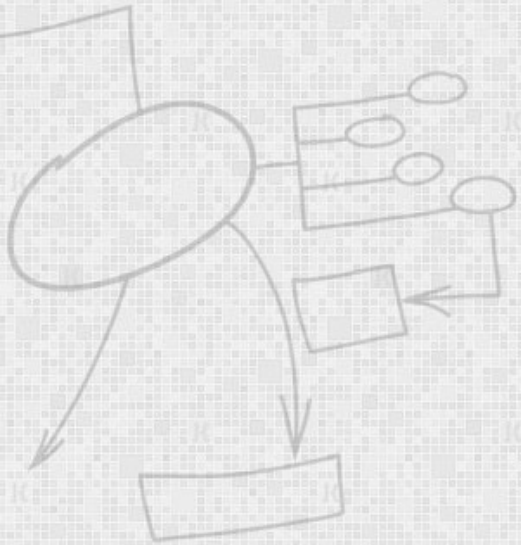
	Huang et al.	Montenegro et al.	Kang et al.	Yusov	Bi et al.	Present thesis
FPS	99.5	30	60	105 (GTX 680)	50	> 150 (GTX 1050 non-Ti)

Table 4.1 Performance comparison

Conclusions

Achievements:

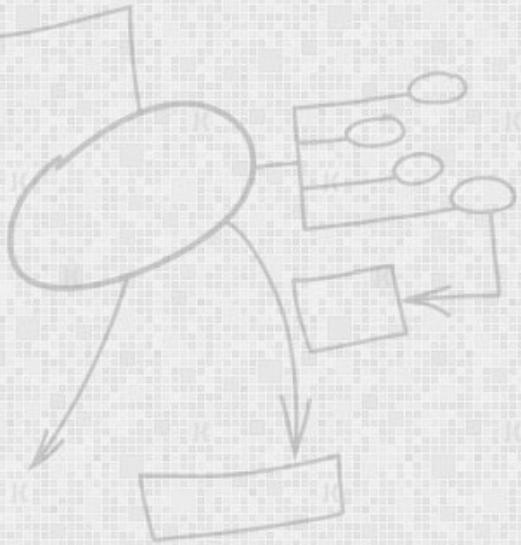
- Optimized noise structure



Conclusions

Achievements:

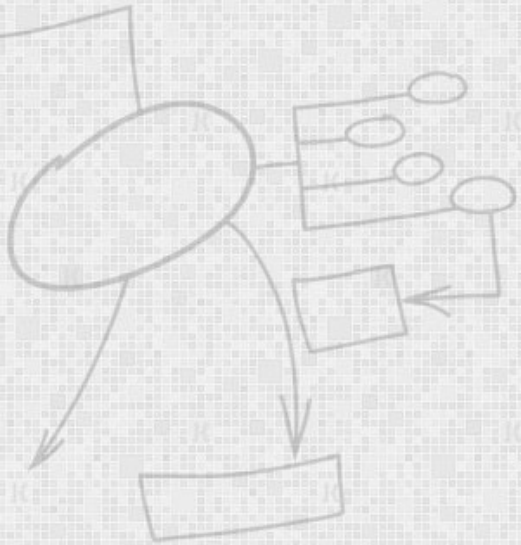
- Optimized noise structure
- Low number of pseudo-spheroids



Conclusions

Achievements:

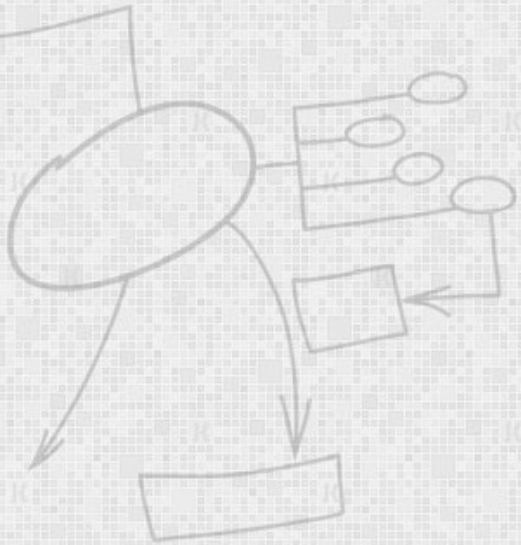
- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes



Conclusions

Achievements:

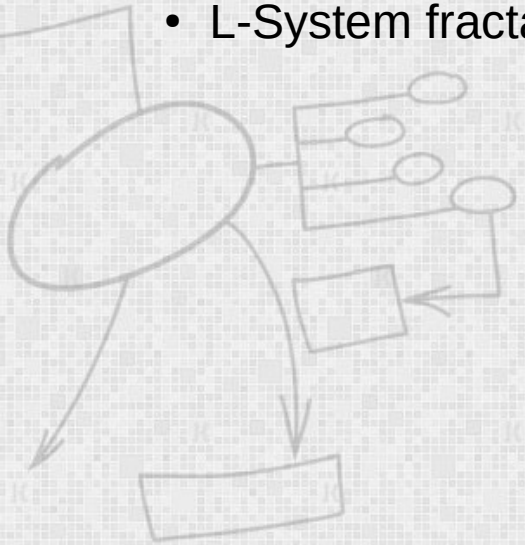
- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes
- Improved Gaussian equations



Conclusions

Achievements:

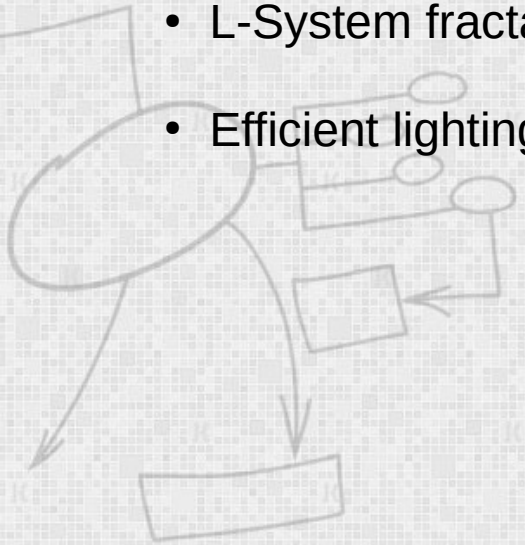
- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes
- Improved Gaussian equations
- L-System fractal cloud generation



Conclusions

Achievements:

- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes
- Improved Gaussian equations
- L-System fractal cloud generation
- Efficient lighting system



Conclusions

Achievements:

- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes
- Improved Gaussian equations
- L-System fractal cloud generation
- Efficient lighting system
- Fast light pre-calculation (NDT algorithm)

Conclusions

Achievements:

- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes
- Improved Gaussian equations
- L-System fractal cloud generation
- Efficient lighting system
- Fast light pre-calculation (NDT algorithm)
- Smooth 3D mesh clouds

Conclusions

Achievements:

- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes
- Improved Gaussian equations
- L-System fractal cloud generation
- Efficient lighting system
- Fast light pre-calculation (NDT algorithm)
- Smooth 3D mesh clouds
- GPGPU cloud dynamics and morphing

Conclusions

Achievements:

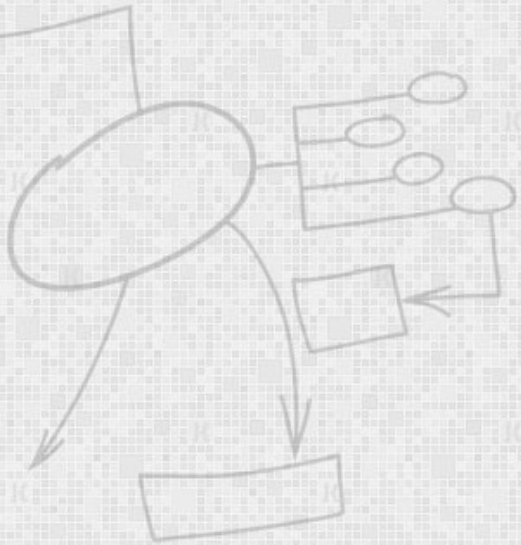
- Optimized noise structure
- Low number of pseudo-spheroids
- Improved bounding boxes
- Improved Gaussian equations
- L-System fractal cloud generation
- Efficient lighting system
- Fast light pre-calculation (NDT algorithm)
- Smooth 3D mesh clouds
- GPGPU cloud dynamics and morphing by using CUDA

Thesis

The initial hypothesis is confirmed and these algorithms are good candidates for applications requiring an optimum balance between realism and performance to be applied in the entry-level graphics industry.

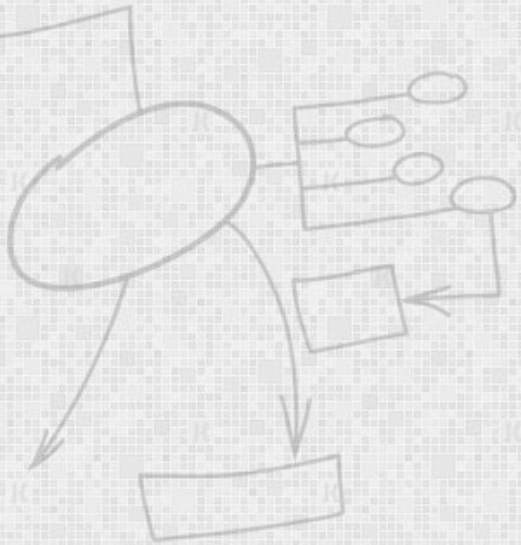
Future work

- Application of space partitioning algorithms (Octrees, K-D trees, etc.)



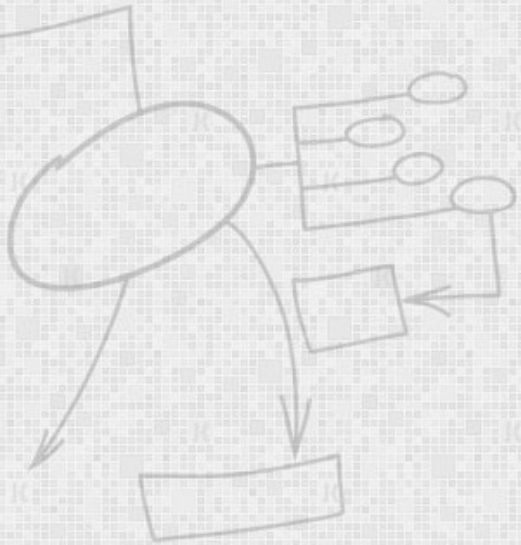
Future work

- Application of space partitioning algorithms (Octrees, K-D trees, etc.)
- A precipitation model



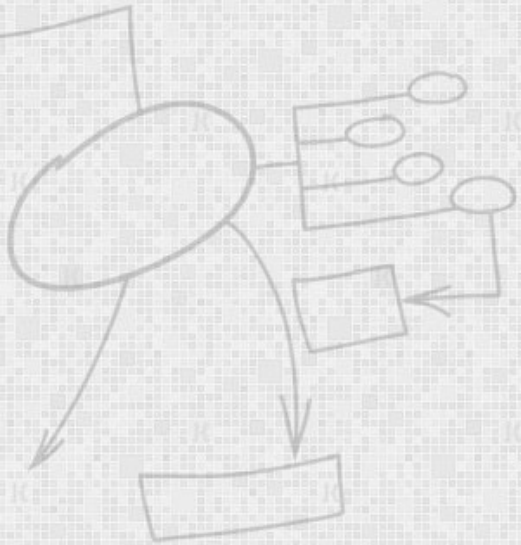
Future work

- Application of space partitioning algorithms (Octrees, K-D trees, etc.)
- A precipitation model
- Shadow mapping



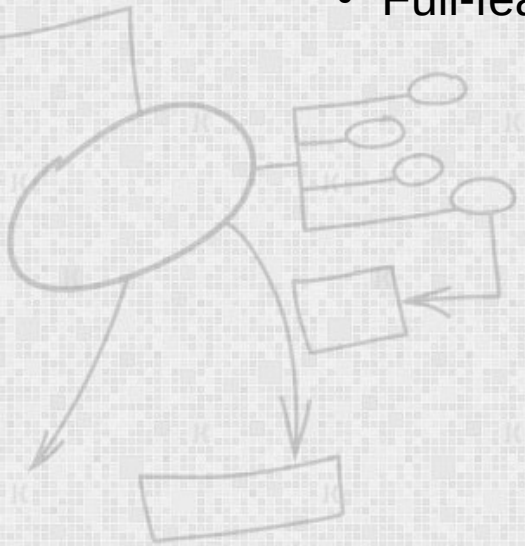
Future work

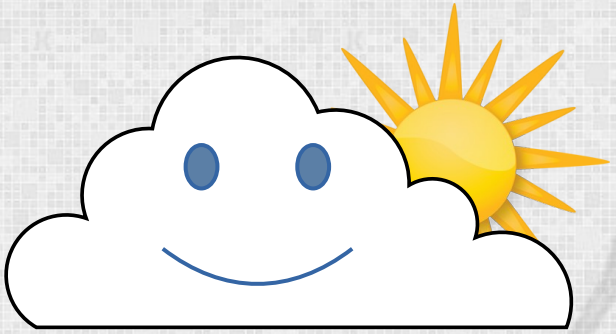
- Application of space partitioning algorithms (Octrees, K-D trees, etc.)
- A precipitation model
- Shadow mapping
- Light beams



Future work

- Application of space partitioning algorithms (Octrees, K-D trees, etc.)
- A precipitation model
- Shadow mapping
- Light beams
- Full-featured cloud deformation



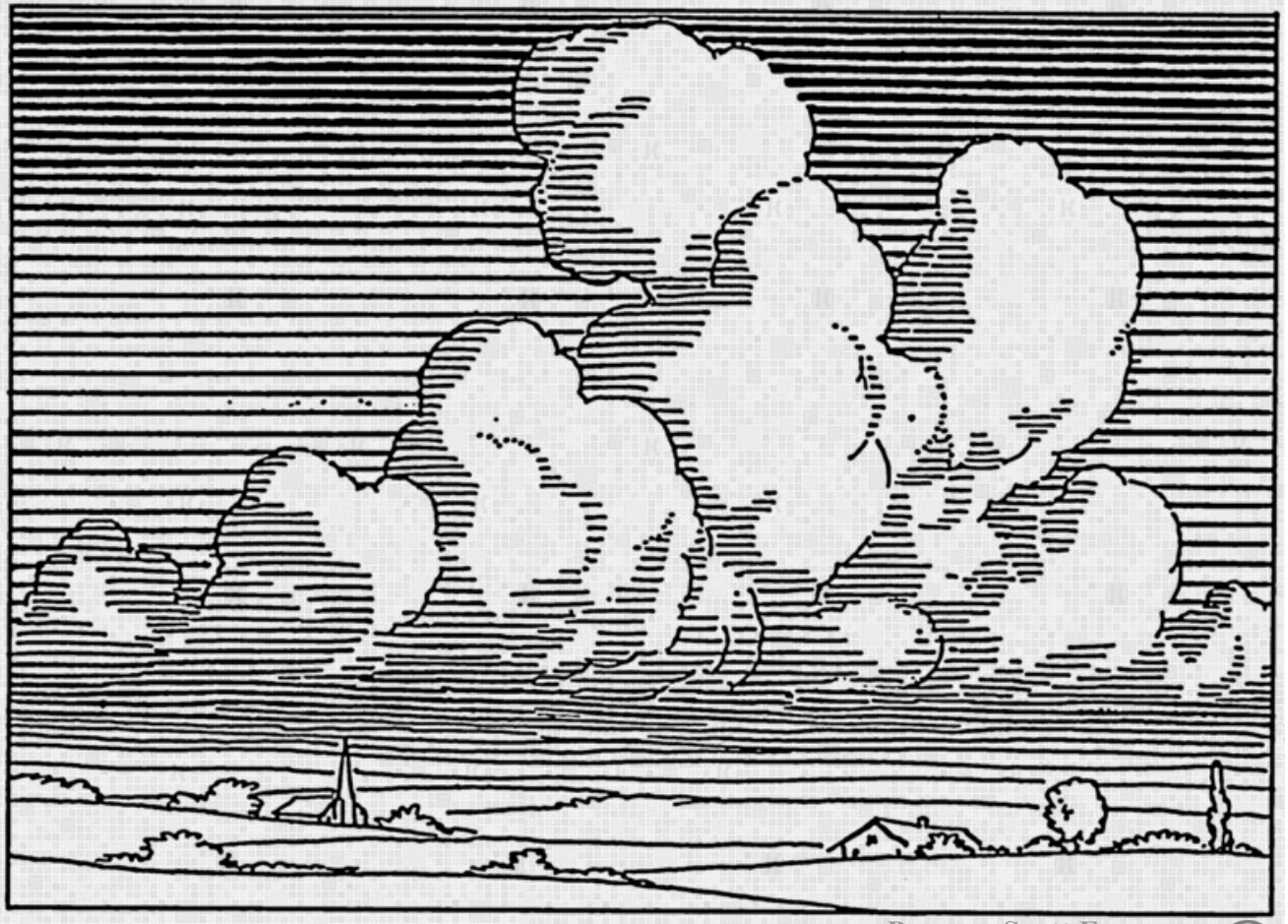


¡Muchas gracias!

UNED

Escuela Internacional de Doctorado

EIDUNED



Pearson Scott Foresman

High-Performance Algorithms for Real-Time GPGPU Volumetric Cloud Rendering from an Enhanced Physical-Math Abstraction Approach

Carlos Jiménez de Parga Bernal - Quirós

Director: José A. Cerrada Somolinos

Co-director: Sebastián R. Gómez Palomo

Bibliography and references

[ASW13]
M. Ament, F. Sadlo, and D. Weiskopf. “Ambient Volume Scattering”. In: IEEE Transactions on Visualization and Computer Graphics 19.12 (Oct. 2013), pp. 2936–2945. doi: 10.1109/TVCG.2013.129.

[Bi+16]
S. Bi et al. “3-Dimensional Modeling and Simulation of the Cloud Based on Cellular Automata and Particle”. In: ISPRS international journal of geo-information 5.6 (June 2016), p. 86. doi: 10.3390/ijgi5060086.

[Bou+08]
A. Bouthors. “Realistic rendering of clouds in real-time”. PhD thesis. Université Joseph Fourier, 2008

[Cer+05]
E. Cerezo et al. “A survey on participating media rendering techniques”. In: Springer Verlag 21.5 (June 2005), pp. 303–328. doi: 10.1007/s00371-005-0287-1

[Har02]
M.J. Harris. “Real-time Cloud Rendering for Games”. In: Proc. Game Developers Conference. 2002.

[Har03]
M.J. Harris. “Real-Time Cloud Simluation and Rendering”. PhD thesis. Chapel Hill, 2003

Bibliography and references

[Häc06] H. Häckel. Clouds: Identification guide. Editorial Omega, 2006

[Hor+05]
L. Horng-Shyang et al. “Fast rendering of dynamic clouds”. In: Computer & Graphics 29.1 (Feb. 2005), pp. 29–40. doi: 10.1016/j.cag.2004.11.005.

[Hua+08]
B. Huang et al. “Study and Implement About Rendering of Clouds in Flight Simulation”. In: Proc. IEEE ’08. 2008, pp. 1250–1254. doi: 10.1109/ICALIP.2008.4590228.

[Jar+08]
W. Jarosz et al. “Radiance Caching for Participating Media”. In: ACM Transactions on Graphics (TOG) 27.1 (Mar. 2008). doi: 10.1145/1330511.1330518

[JG18]
C. Jiménez de Parga and S.R. Gómez Palomo. “Efficient Algorithms for Real-Time GPU Volumetric Cloud Rendering with Enhanced Geometry”. In: Symmetry 10.4 (2018), p. 125

[Kal+17] S. Kallweit et al. “Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks”. In: ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2017) (Sept. 2017). doi: 10.1145/3130800.3130880.

Bibliography and references

[Kni+02]
J. Kniss et al. “Interactive translucent volume rendering and procedural modeling”. In: IEEE Visualization, 2002. VIS 2002. IEEE. 2002, pp. 109–116

[KPK15]
S.Y. Kang, K.C. Park, and K.l Kim. “Real-Time Cloud Modeling and Rendering Approach Based on L-system for Flight Simulation”. In: International Journal of Multimedia and Ubiquitous Engineering 10.6 (June 2015), pp. 395–406.

[Max95]
N. Max. “Optical Models for Direct Volume Rendering”. In: IEEE Transactions on Visualization and Computer Graphics 1.2 (June 1995), pp. 99–108. doi: 10.1109/2945.468400

[MB15]
K. Mukina and A. Bezgodov. “The Method for Real-time Cloud Rendering”. In: Procedia Computer Science. Vol. 66. 2015, pp. 697–704. doi: 10.1016/j.procs.2015.11.079

[Mon+17]
A. Montenegro et al. “A new method for modeling clouds combining procedural and implicit models”. In: 2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). IEEE. 2017.

Bibliography and references

[Nar+06]
S.G. Narasimhan et al. “Acquiring Scattering Properties of Participating Media by Dilution”. In: ACM Transactions on Graphics (TOG) 25.3 (July 2006), pp. 1003–1012. doi: 10.1145/1141911.1141986.

[Smi02]
B. Smits. “Efficient bounding box intersection”. In: Ray Tracing News 15.1 (Oct. 2002), p. 1

[Sta03]
J. Stam. “Real-time fluid dynamics for games”. In: Proceedings of the game developer conference. Vol. 18. 2003, p. 25

[WBC08]
J. Wither, A. Bouthors, and M.P. Can. “Rapid sketch modeling of clouds”. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM) (June 2008), pp. 113–118

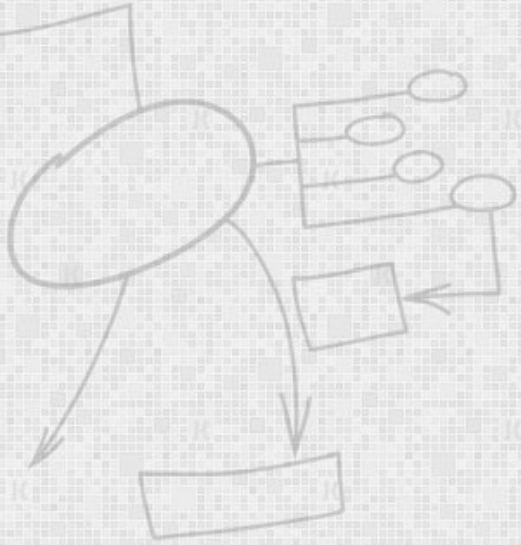
[Wil+05]
A. Williams et al. “An efficient and robust ray-box intersection algorithm”. In: SIGGRAPH ’05 ACM SIGGRAPH 2005. 9. 2005, pp. 1–4. doi: 10.1145/1198555.1198748

Bibliography and references

[Yus14]
E. Yusov. “Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics”. In: High-Performance Rendering of Realistic Cumulus Clouds Using Pre-computed Lighting. 2014, pp. 127–136. doi: 10.2312/hpg.20141101.

[YW11]
C.M. Yu and C.M. Wang. “An Effective Framework for Cloud Modeling, Rendering, and Morphing.” In: J. Inf. Sci. Eng. 27.3 (2011), pp. 891–913

[Zho+08]
K. Zhou et al. “Real-time smoke rendering using compensated ray marching”. In: SIGGRAPH ’08. 36. 2008, pp. 1–12. doi: 10.1145/1399504.1360635



Copyright notice

- Example of a mouse skull (CT) rendering using the shear warp algorithm by Lackas CC BY-SA 3.0.
- Cloud taxonomy is CC BY-SA 3.0 by Valentin de Bruyn / Coton.