WAKES, EXPLOSIONS AND LIGHTING: INTERACTIVE WATER SIMULATION IN 'ATLAS'

Mark Mihelich (Grapeshot Games)
Tim Tcheblokov (NVIDIA)

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Introduction to ATLAS

ATLAS

Massive multiplayer first- and third-person fantasy pirate adventure

ATLAS

Massive multiplayer first- and third-person fantasy pirate adventure

Agenda

Sailing in Atlas is important part of the gameplay

Many things must be done right:

- Simulation of sea states
- Physics of buoyancy
- Rendering the seas
- Interactive features: wakes and explosions

This is the agenda for today’s talk

Simulation of sea states

Basics

Spectrum based approach
Evolved from “Simulating Ocean Water” by J. Tessendorf:

- Generate spectrum
  - Evolve spectrum in frequency domain
  - Transform data to spatial domain using inverse FFTs
  - Rinse and repeat

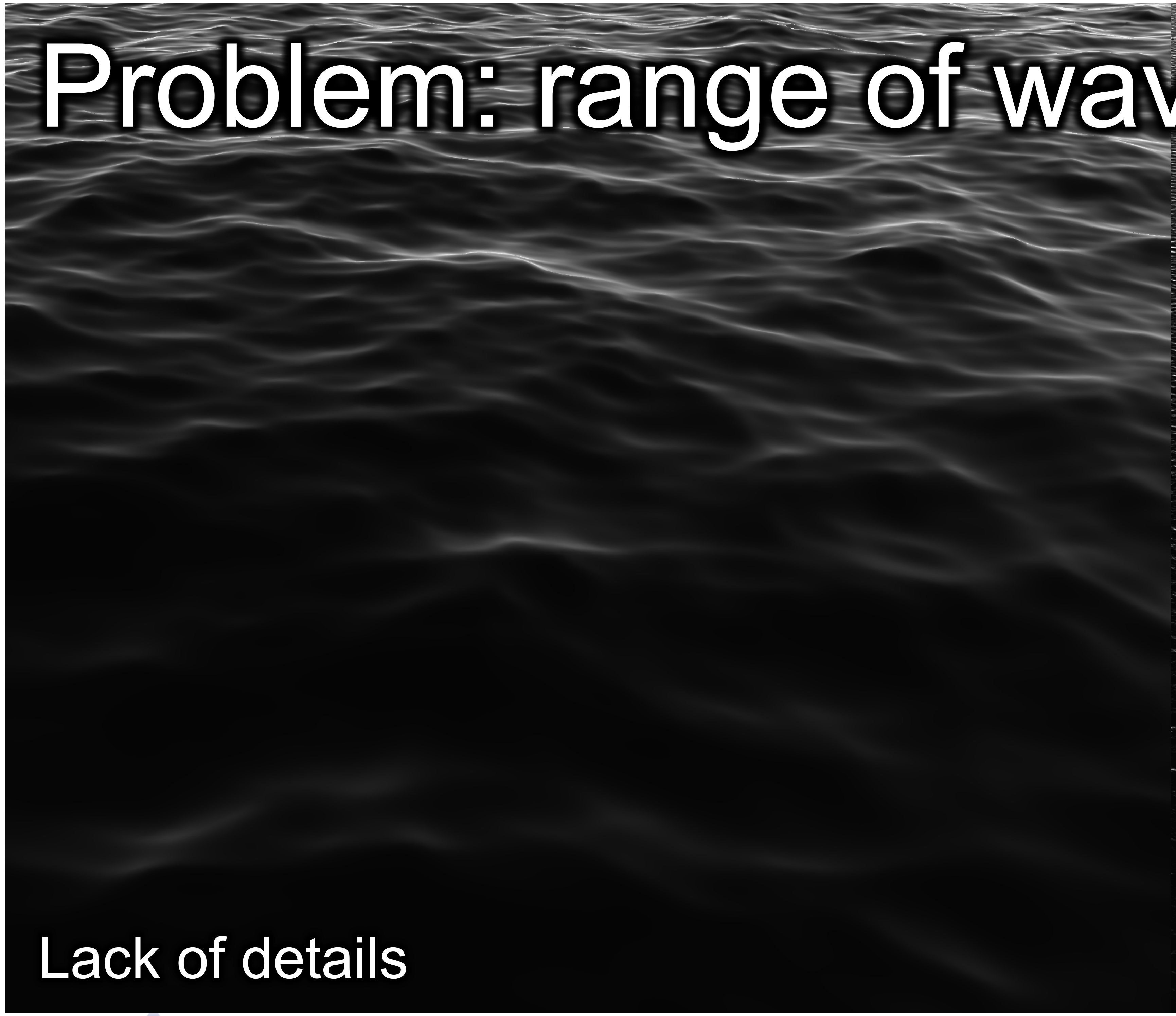
Basics

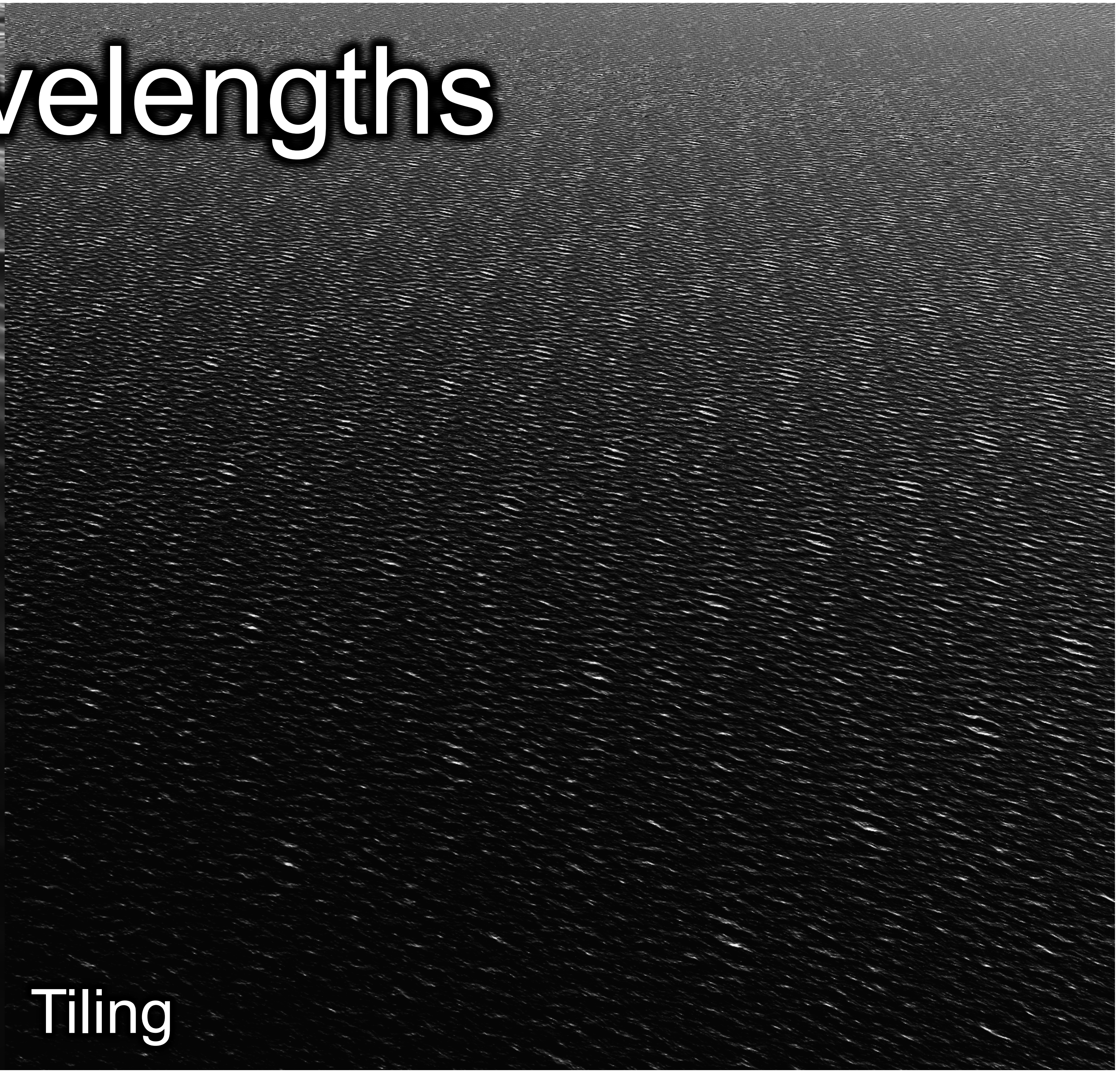
Good properties:

- Only depends on absolute time and spectrum parameters
  - Can be simulated independently on server and clients
  - Or all the servers in the grid
    - Seamless transition
- Results are tiles with periodic displacement data
  - Can cover infinite areas seamlessly

Nice and easy to use, but not good enough for us!

Problem: range of wavelengths





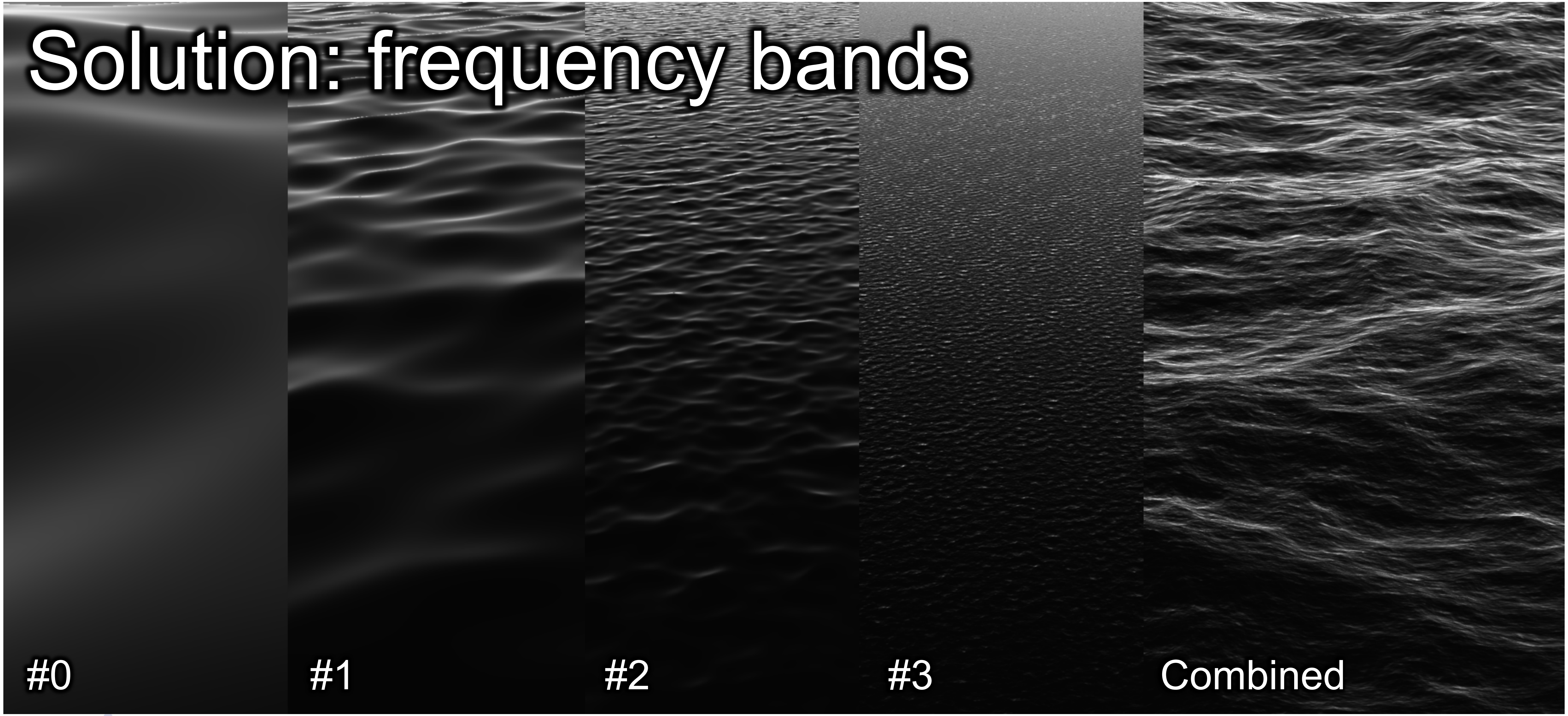
Solution: frequency bands

Large FFTs are expensive $\left(O(N^2 \log N)\right)$, small FFTs produce visible tiles

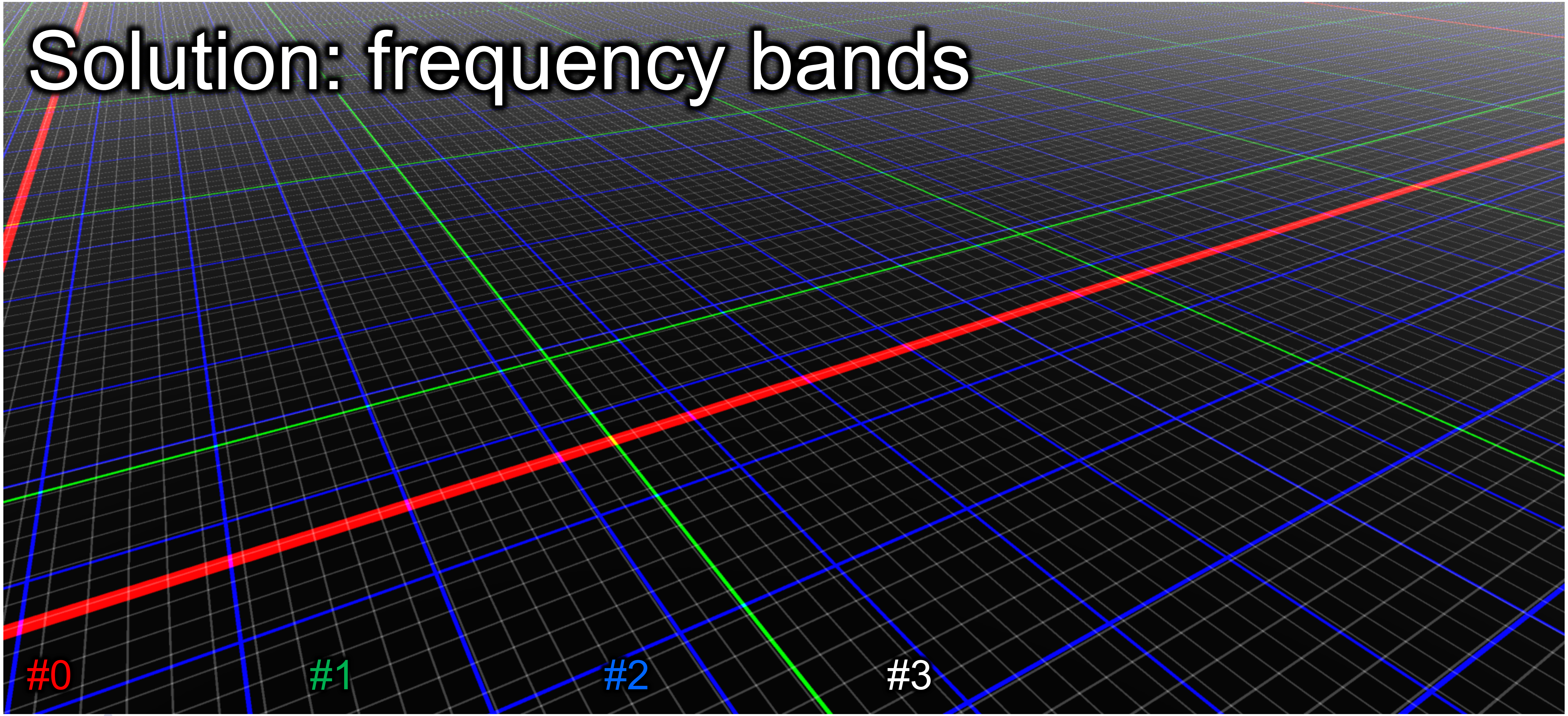
Our solution:

- Use small FFTs
- Split spectrum to 4 frequency bands based on wave lengths
  - Evolve all 4 bands at the same time
  - Convert 4 bands to spatial domain with inverse FFTs
  - Postprocess results to get BRDF for PBR
  - Recombine results from 4 bands in the shaders

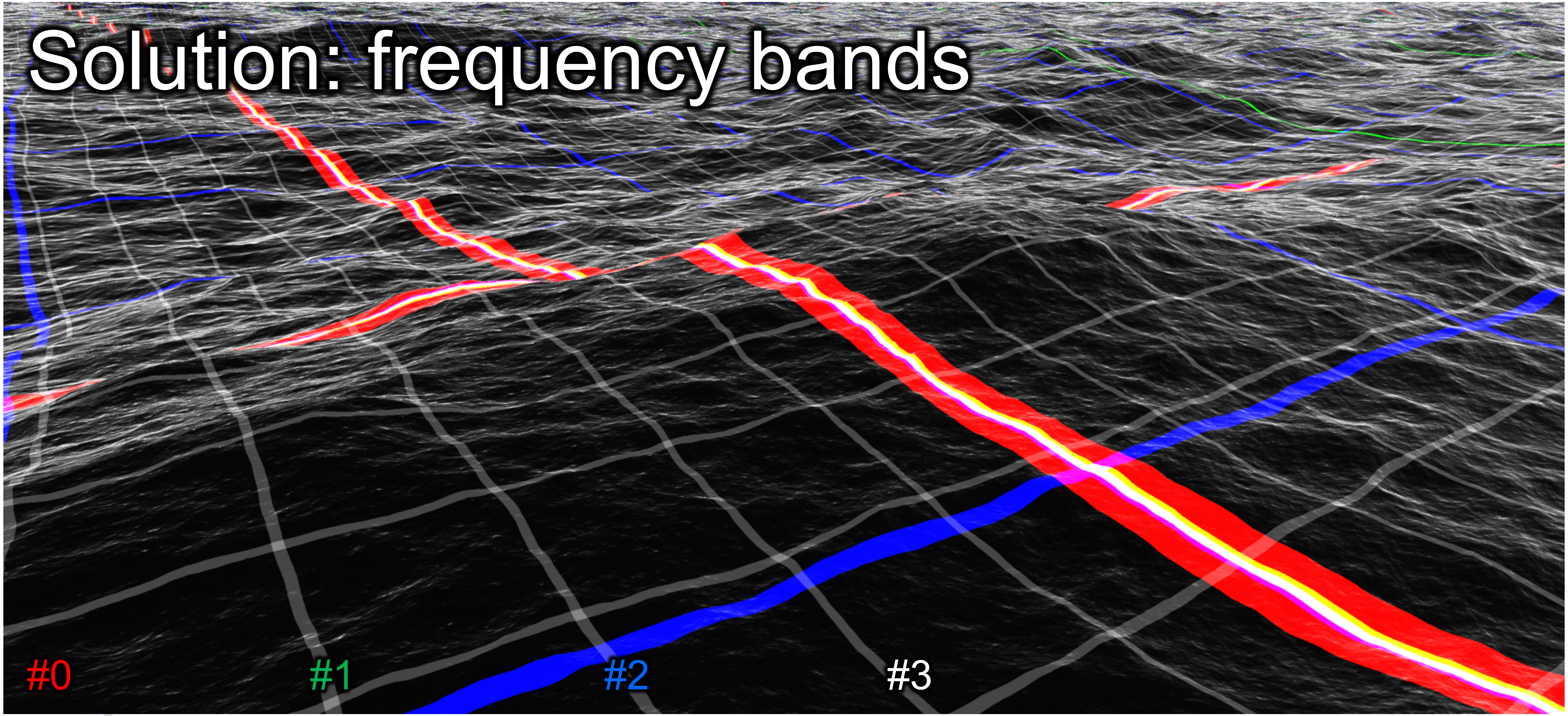
Solution: frequency bands



Solution: frequency bands



Solution: frequency bands



Problem: Phillips spectrum

Phillips spectrum is not the best and is not customizable enough

Seas are never fully evolved

- 100MPH wind over a puddle?

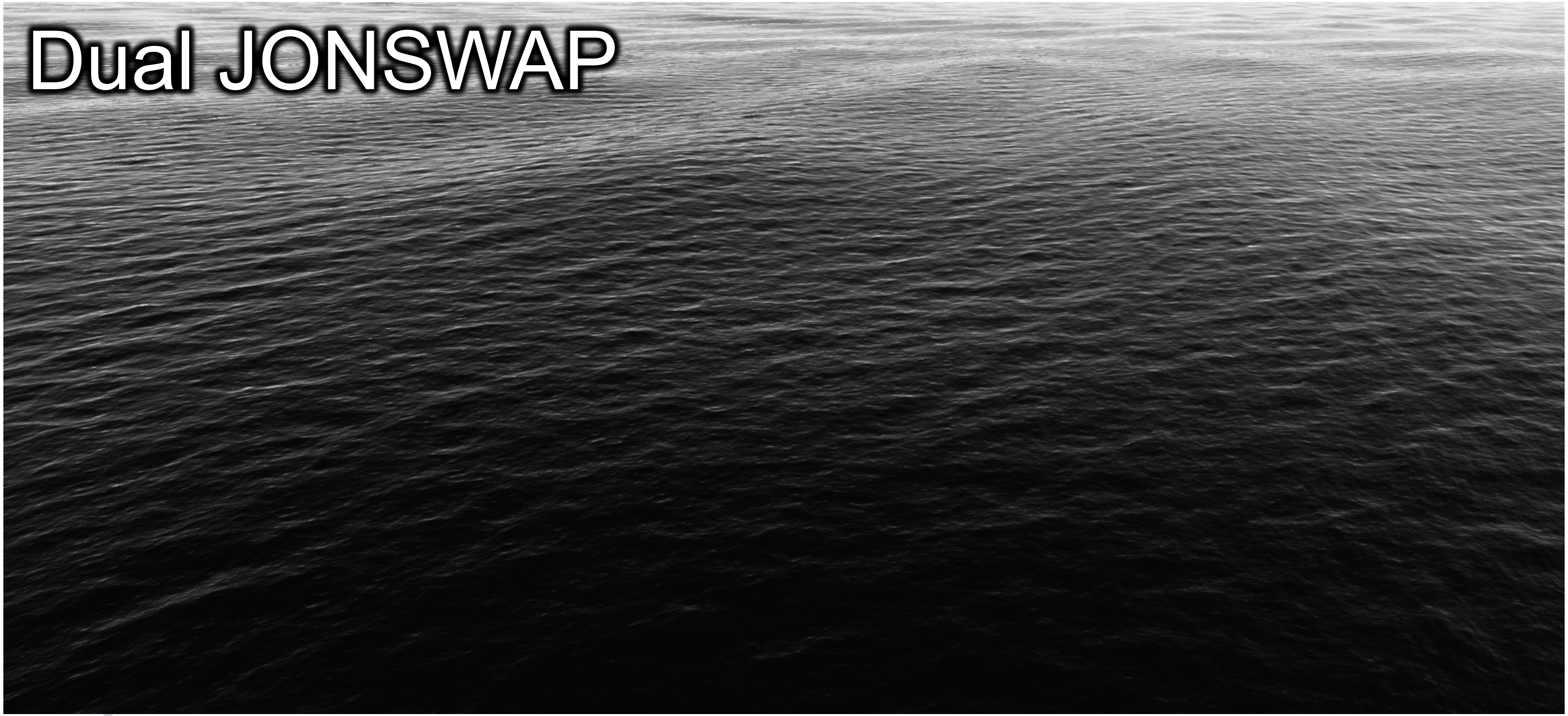
Dual JONSWAP

Solution:

- Dual fetch limited JONSWAP spectra
  - Small/medium local wind waves + large smooth swell waves
  - A lot of artistic control while staying physically correct

JOint North Sea WAve Project, 1973

Dual JONSWAP



Dual JONSWAP

A lot of artistic control:

- Wind speed
- Wind direction
- Wind fetch
- Spectrum peaking
- Directional distribution
- Override amplitude
- Low pass filter

Physics of buoyancy

Physics of buoyancy

Archimedes Principle

$$F = (\rho_f - \rho_g) \cdot g \cdot V$$

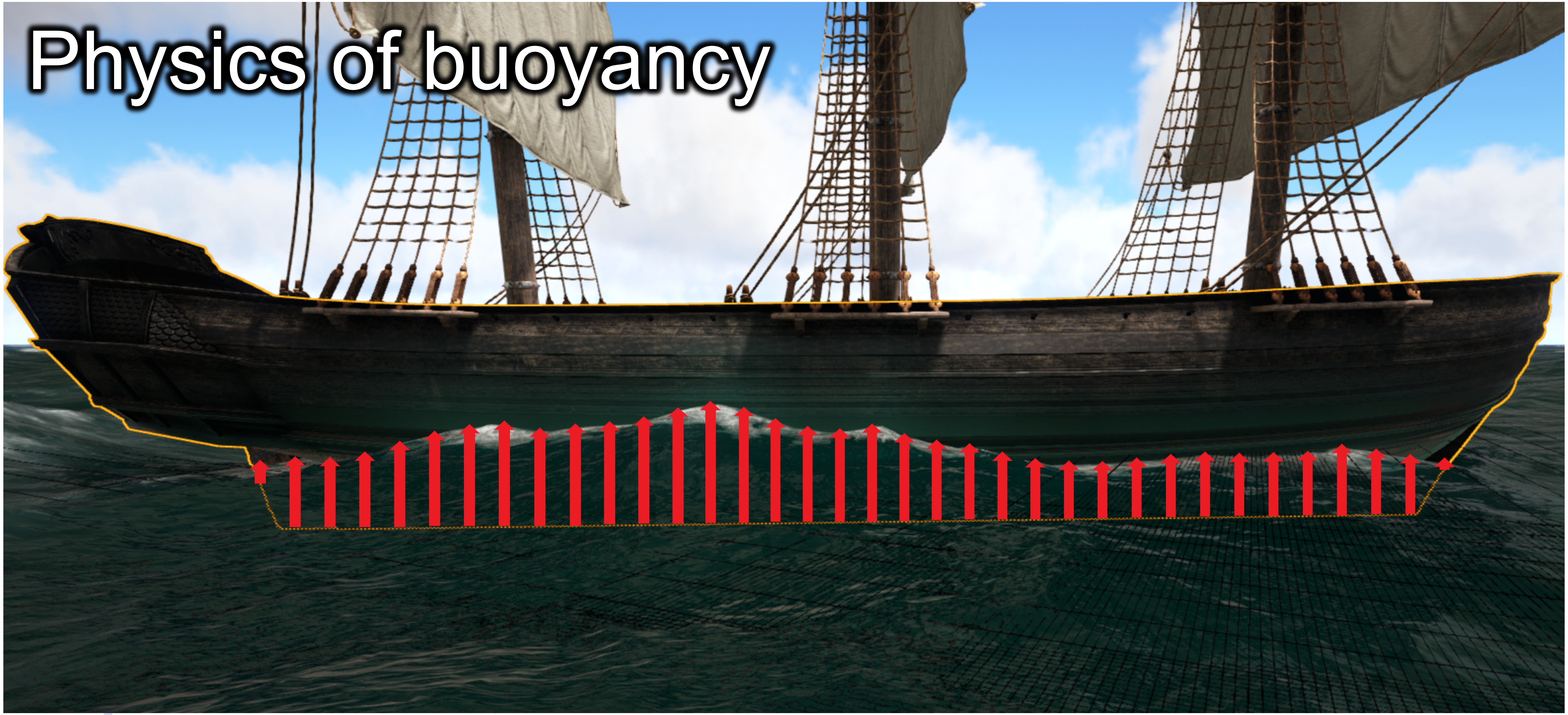
F – Buoyancy force
ρ_f – Density of the fluid (water)
ρ_g – Density of the body (our ship)
g – Acceleration due to gravity
V – Displaced volume of fluid

Physics of buoyancy

- Take *n* discrete sea surface displacement samples along the hull of the boat
- Each sample represents a top-down cross-section of the hull
- Each column volume assumed to have uniform density
- We can calculate a buoyancy force for each sample separately and apply individual forces back into the boat's rigid body physics simulation

Physics of buoyancy

Physics of buoyancy



Physics of buoyancy

```
void ApplyBuoyancy(RigidBody* Boat, Array<Vec3>& SamplePoints)
{
   float UnitForce = (kWaterDensity - Boat->density) * kSampleArea;

   for(Vec3& SamplePoint : SamplePoints)
   {
      float WaterHeight = GetWaterHeightAtPoint(SamplePoint);
      float Displacement = max(0, WaterHeight - SamplePoint.Z);
      Vec3 BuoyancyForce = -vGravity * Displacement * UnitForce;

      Boat->ApplyForceAtLocation(SamplePoint, BuoyancyForce);
   }
}
```

Physics of buoyancy

Issues with physically-based method:

- Discrete wave height samples noisy in time domain
- Server simulation often runs at < 10Hz
- Relying on forces and rigid body dynamics adds latency and instability
- We want epic wave size without making players sick

Physics of buoyancy

Our solution:

- We still use our *n* discrete wave height samples
- Samples used as input to plane fitting algorithm
  - David Eberly's Geometric Tools contain a useful plane fitting implementation: https://www.geometrictools.com/Samples/Mathematics.html#SymmetricEigensolver3x3
- Use calculated plane for target ship transform
- Apply spring to filter noise and mimic physics of buoyancy

Physics of buoyancy

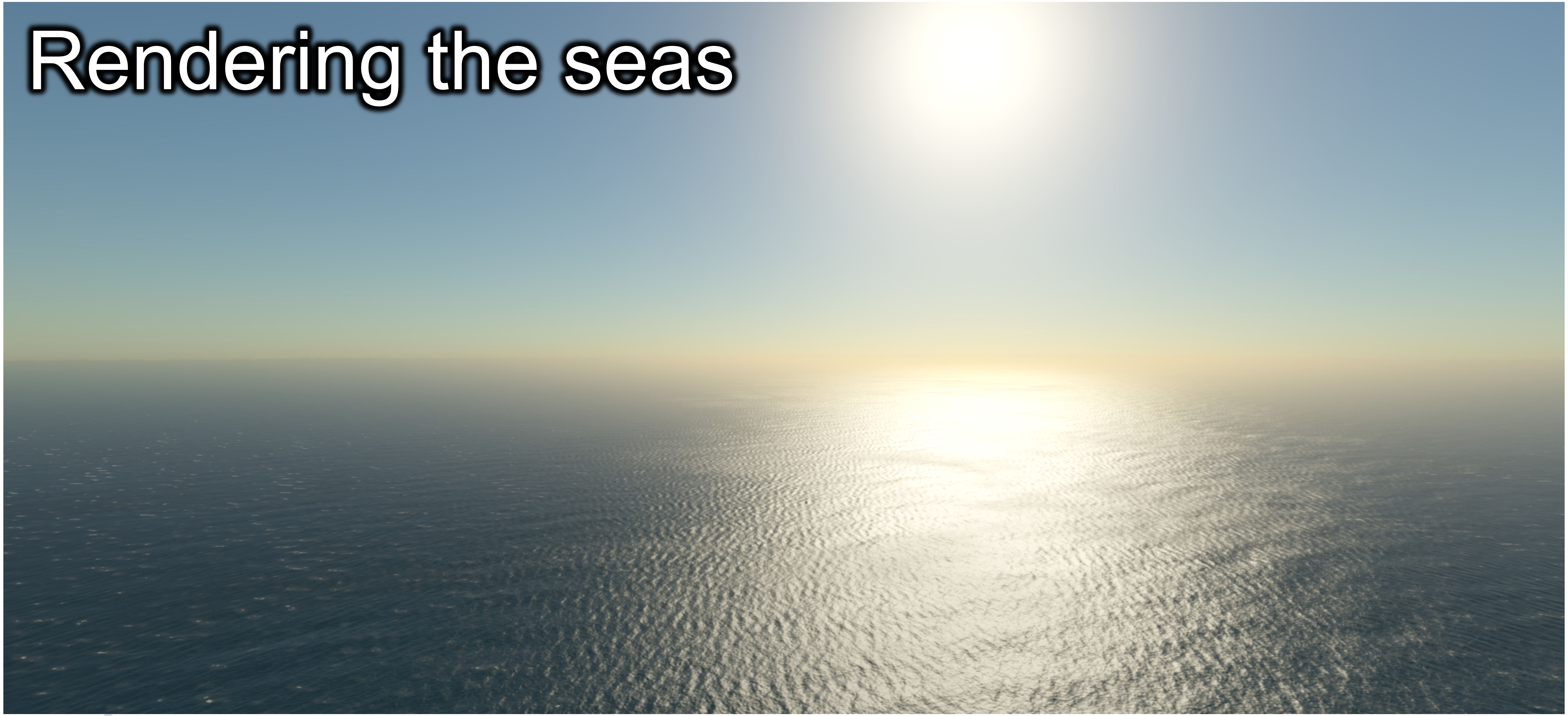
Note about calculating wave height:

- Wave simulation outputs 3D displacements relative to an imaginary plane
- We want to convert this displacement back into world space
- An iterative approximation is required

Physics of buoyancy

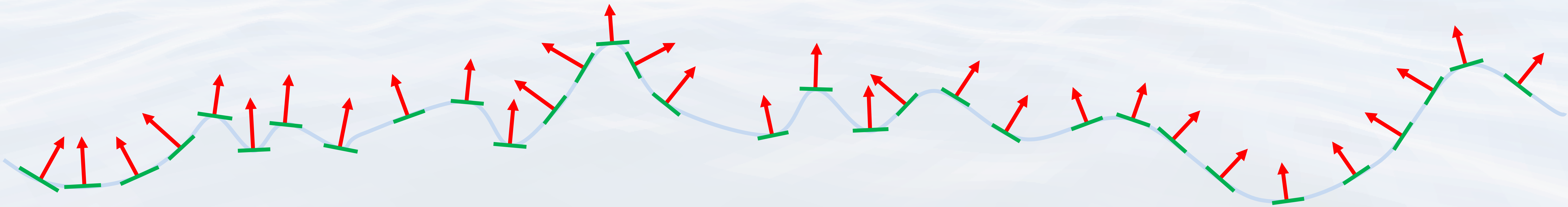
```
float GetWaterHeightAtPoint(Vec3& SamplePoint)
{
   Vec3 Disp = Vec3(0,0,0);
   for(int k =0; k < NUM_ITERATIONS; k++)
   {
      Disp = GetDisplacement(SamplePoint - Disp);
   }
   return Disp.Z;
}
```

Rendering the seas



Sea as microfacet surface

Waves, microfacets and micronormals:



Sea as microfacet surface

Rendering equation

Rendering equation:

$$L_{to\_eye} = L_{scatter} + \int_{\Omega} L_{sun} \cdot f_r \cdot \cos\theta \, \mathrm{d}\omega + \int_{\Omega} L_{env} \cdot f_r \cdot \cos\theta \, \mathrm{d}\omega$$

f_r – BRDF - surface reflectance depending on incoming and outgoing angles

BRDF

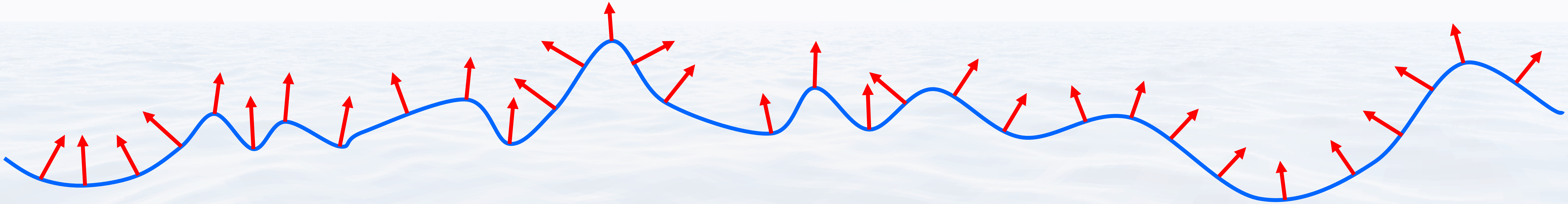
Microfacet BRDF model:

$$f_r = \frac{F \cdot D \cdot G}{4 \cdot \cos\theta_i \cdot \cos\theta_o}$$

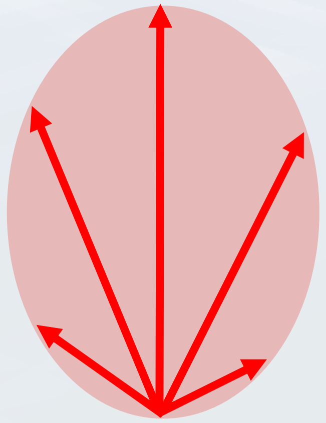
F – Fresnel reflectance
D – Normal distribution function
G – Masking function

NDF

Micronormals:



Micronormal distribution:



NDF

We use Beckmann distribution:

$$D(\omega_n) = \frac{P_{22}(\bar{n})}{\left(\omega_n \cdot \omega_g\right)^4} \qquad P_{22}(\bar{n}) = \frac{1}{\pi \alpha_x \alpha_y} \exp\left(-\frac{x_{\bar{n}}^2}{\alpha_x^2} - \frac{y_{\bar{n}}^2}{\alpha_y^2}\right)$$

$P_{22}(\bar{n})$ – 2D PDF, probability of finding the facet with normal $\bar{n}$ or $(x_{\bar{n}}, y_{\bar{n}})$ slopes

α_x , α_y – surface roughness along X and Y axis

We set $\left(\omega_n \cdot \omega_g\right)^4 = 1$: ω_n (mesonormal) equals ω_g (macronormal) for us.

Surface moments

We can write the PDF in terms of moments (LEADR mapping):

$$P_{22}(\bar{n}) = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(\bar{n} - E[\bar{n}])^t \Sigma^{-1} (\bar{n} - E[\bar{n}])\right), \qquad \Sigma = \begin{bmatrix} \delta_x^2 & c_{xy} \\ c_{xy} & \delta_y^2 \end{bmatrix}$$

Σ is the covariance matrix based on slope moments:

$$\delta_x^2 = E\left[x_{\bar{n}}^2\right] - E^2[x_{\bar{n}}], \qquad \delta_y^2 = E\left[y_{\bar{n}}^2\right] - E^2[y_{\bar{n}}], \qquad c_{xy} = E[x_{\bar{n}} y_{\bar{n}}] - E[x_{\bar{n}}]E[y_{\bar{n}}]$$

PDF is now written in terms of first and second moments
that allow linear operators and can be precalculated and stored in textures

Mipmapping and combination are the linear operators we will gladly use!

Calculating moments

Reverse FFT steps provide displacements

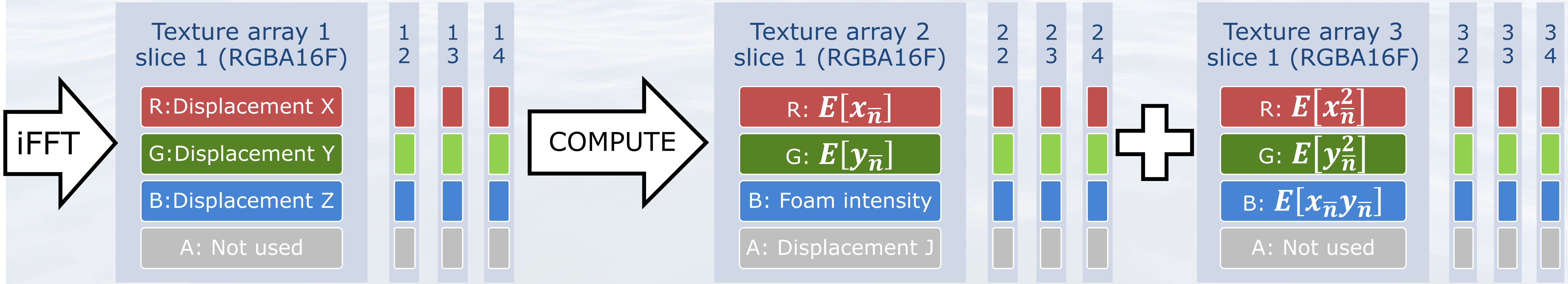
Using them, we calculate:

- First order moments $E[x_{\bar{n}}]$, $E[y_{\bar{n}}]$ or slopes of the surface
- Second order moments $E\left[x_{\bar{n}}^2\right]$, $E\left[y_{\bar{n}}^2\right]$ or squares of slopes
- Covariance $E[x_{\bar{n}} y_{\bar{n}}]$, or $E[x_{\bar{n}}] * E[y_{\bar{n}}]$

...and store to textures

Calculating moments

The data we update on every simulation step is the following:



Summing moments

Combining first order moments:

$$E[x_{\bar{n}}] = E_1[x_{\bar{n}}] + E_2[x_{\bar{n}}]$$
$$E[y_{\bar{n}}] = E_1[y_{\bar{n}}] + E_2[y_{\bar{n}}]$$

Second order moments and covariance:

$$E\left[x_{\bar{n}}^2\right] = E_1\left[x_{\bar{n}}^2\right] + E_2\left[x_{\bar{n}}^2\right] + 2E_1[x_{\bar{n}}]E_2[x_{\bar{n}}]$$
$$E\left[y_{\bar{n}}^2\right] = E_1\left[y_{\bar{n}}^2\right] + E_2\left[y_{\bar{n}}^2\right] + 2E_1[y_{\bar{n}}]E_2[y_{\bar{n}}]$$
$$E[x_{\bar{n}}y_{\bar{n}}] = E_1[x_{\bar{n}}y_{\bar{n}}] + E_2[x_{\bar{n}}y_{\bar{n}}] + E_1[x_{\bar{n}}]E_2[y_{\bar{n}}] + E_1[y_{\bar{n}}]E_2[x_{\bar{n}}]$$

Specular reflection

Integrating specular is analytic:

$$\int_{\Omega} L_{sun} \cdot f_r \cdot cos\,\theta\, \mathrm{d}\omega = \frac{L_{sun} \cdot F(\omega_h, \omega_{sun}) \cdot p_{22}(\omega_h)}{4 \cdot (\omega_n \cdot \omega_{eye}) \cdot (1 + \Lambda(\omega_{sun}) + \Lambda(\omega_{eye}))}$$

ω_{sun} , ω_{eye} , ω_h – sun / eye / half vector direction

ω_n – macronormal, (0,0,1) in our case

Specular reflection

Environment reflection

Can't be integrated analytically:

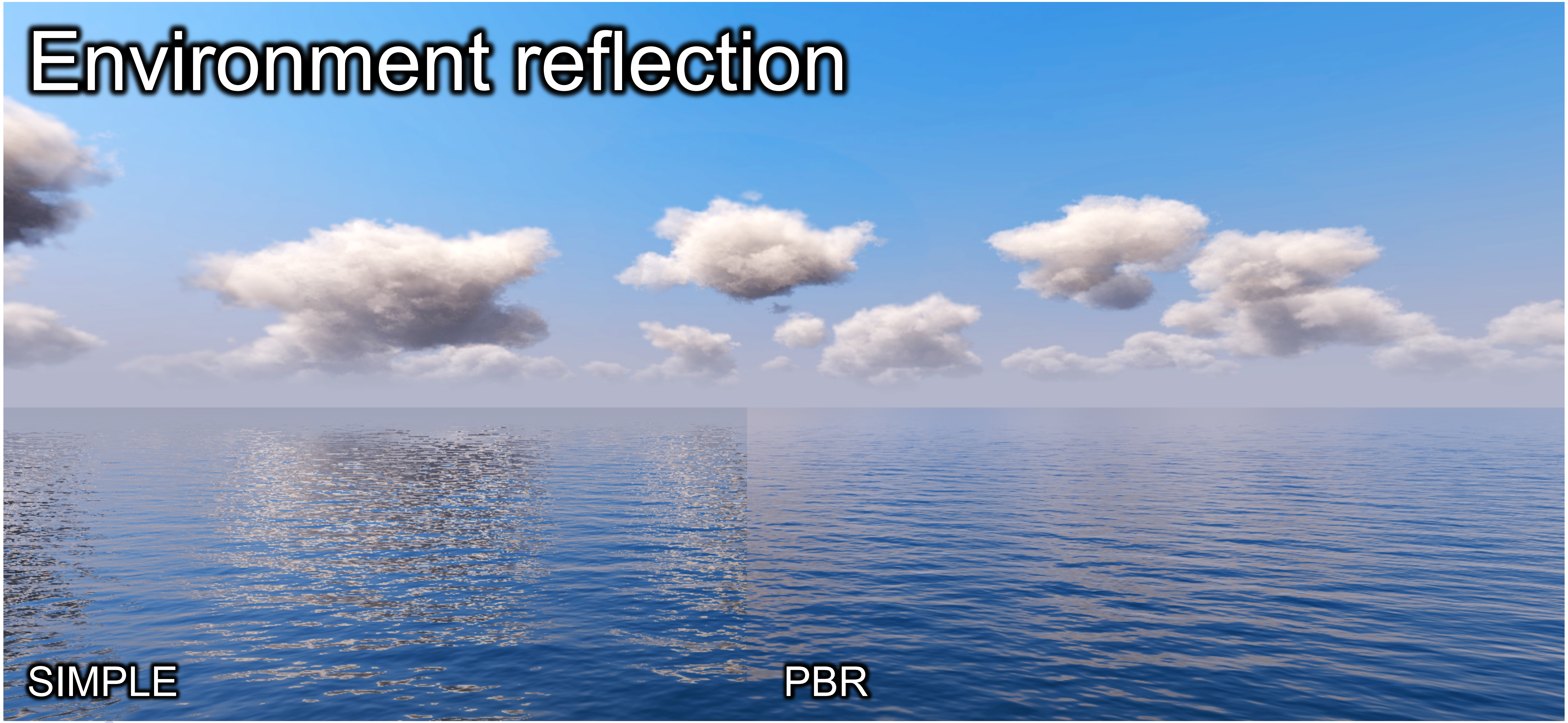
$$\int_{\Omega} L_{env} \cdot f_r \cdot cos\,\theta\,\mathrm{d}\omega$$

Integrate numerically as sum of samples

The math is the same as in LEADR paper, but for sake of performance:

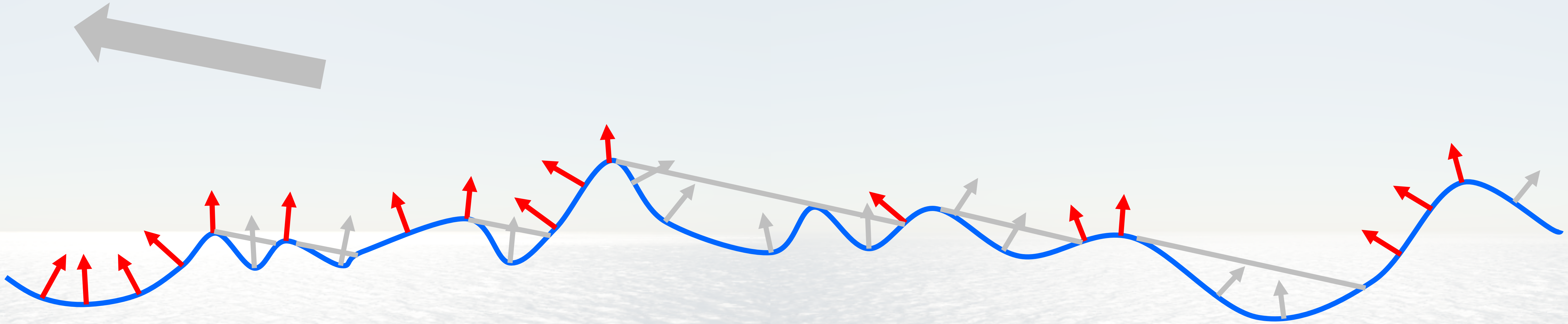
- Small set of samples, 3x3 samples
- No Fresnel for samples, Fresnel for the sum instead
- No masking / shadowing.

Environment reflection



Masking / shadowing

Waves obstruct each other!



Masking / shadowing

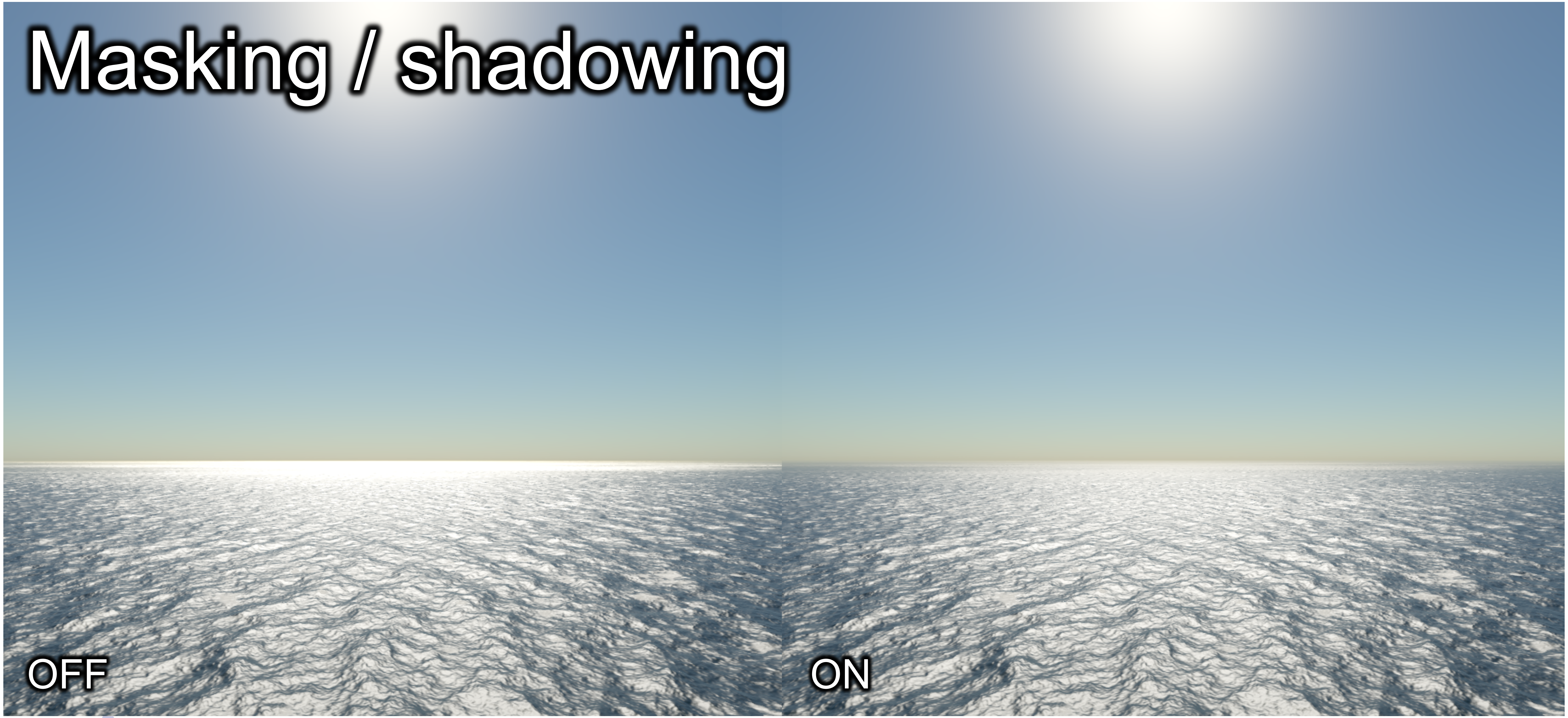
Waves obstruct each other!

Walter's approximation for Smith's masking and shadowing functions:

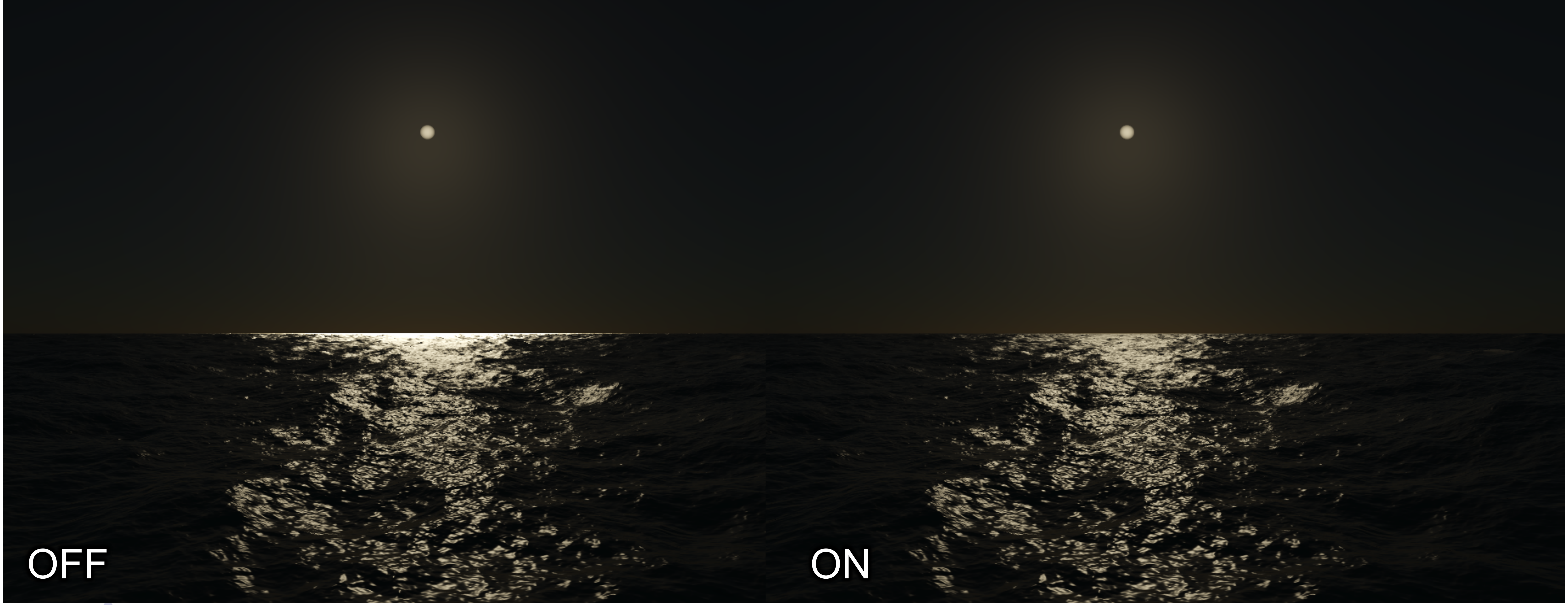
$$G = \frac{1}{1+\Lambda(\omega_i)+\Lambda(\omega_o)}, \qquad \Lambda(\omega) \approx \begin{cases} \frac{1-1.259a+0.396a^2}{3.535a+2.181a^2}, a < 1.6 \\ 0, otherwise \end{cases}, \quad a = \frac{1}{\alpha \tan\theta}$$

ω_i , ω_o – incoming and outgoing light vectors
θ , φ – pairs of angles for those vectors
α – "projected anisotropic roughness" $= \sqrt{\alpha_x{}^2 \cos^2\varphi + \alpha_y{}^2 \sin^2\varphi}$

Masking / shadowing

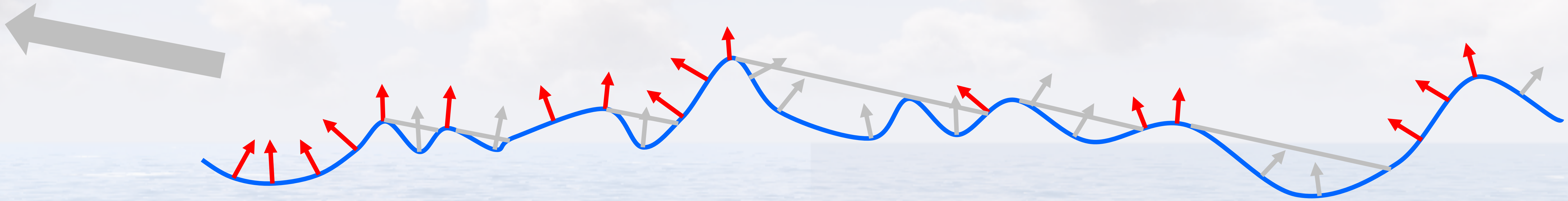


Masking / shadowing

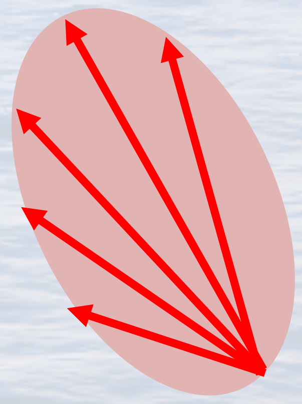


Fresnel reflectance

Visible micronormals:



Visible micronormal distribution:



Fresnel reflectance

Schlick's approximation for the BRDF :

$$F \approx R + (1 - R)\frac{(1 - \cos\theta_v)^{5\exp(-2.69\alpha_v)}}{1 + 22.7\alpha_v^{1.5}}$$

R – $(\eta-1)^2/(\eta+1)^2$

η – air to water refraction factor

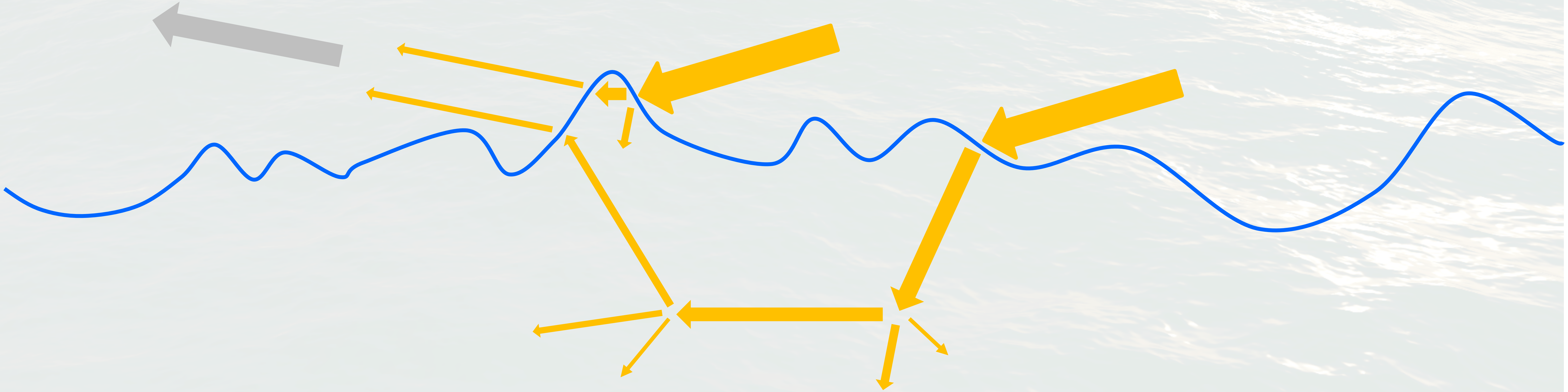
α_v – "projected anisotropic roughness" $= \sqrt{{\alpha_x}^2\cos^2\varphi_v + {\alpha_y}^2\sin^2\varphi_v}$

φ_v, θ_v – angles of the view vector v

Fresnel reflectance

Scattered light

Requires calculating light transport in the water body
Too expensive for real time!



Scattered light

Fake scattering by looking at the probabilities:

$$L_{scatter} = (k_1 H\langle\omega_i \cdot -\omega_o\rangle^4 (0.5 - 0.5(\omega_i \cdot \omega_n))^3 + k_2\langle\omega_o \cdot \omega_n\rangle^2) C_{ss} L_{sun} \cdot \frac{1}{(1 + \Lambda(\omega_i))}$$

$$L_{scatter} \mathrel{+}= k_3\langle\omega_i \cdot w_n\rangle C_{ss} L_{sun} + k_4 P_f C_f L_{sun}$$

H – max(0,wave height), ω_i , ω_o , ω_h – sun / eye / half vector direction

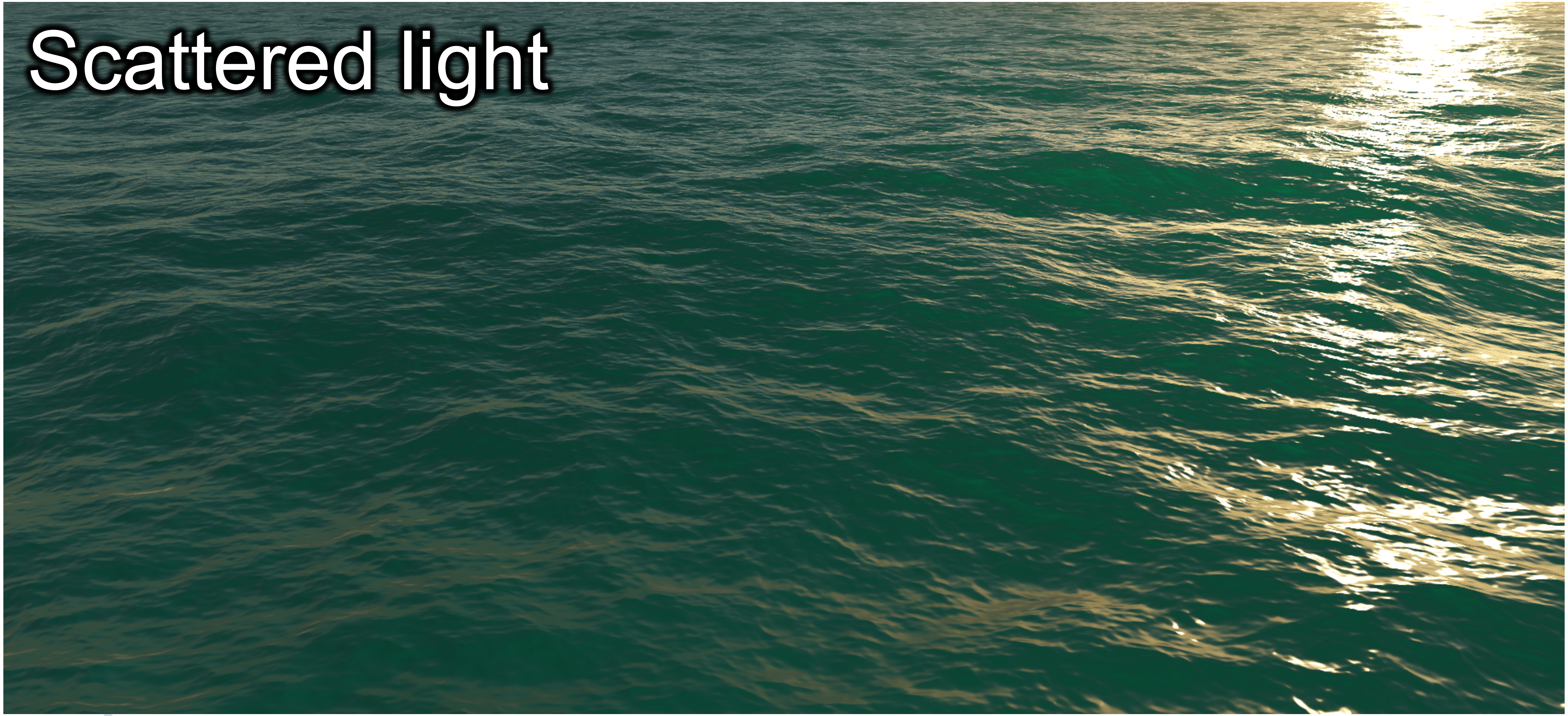
k_1, k_2, k_3, k_4 – tweaking multipliers controlled by artists

C_{ss} , C_f – water scatter color and air bubbles color, controlled by artists

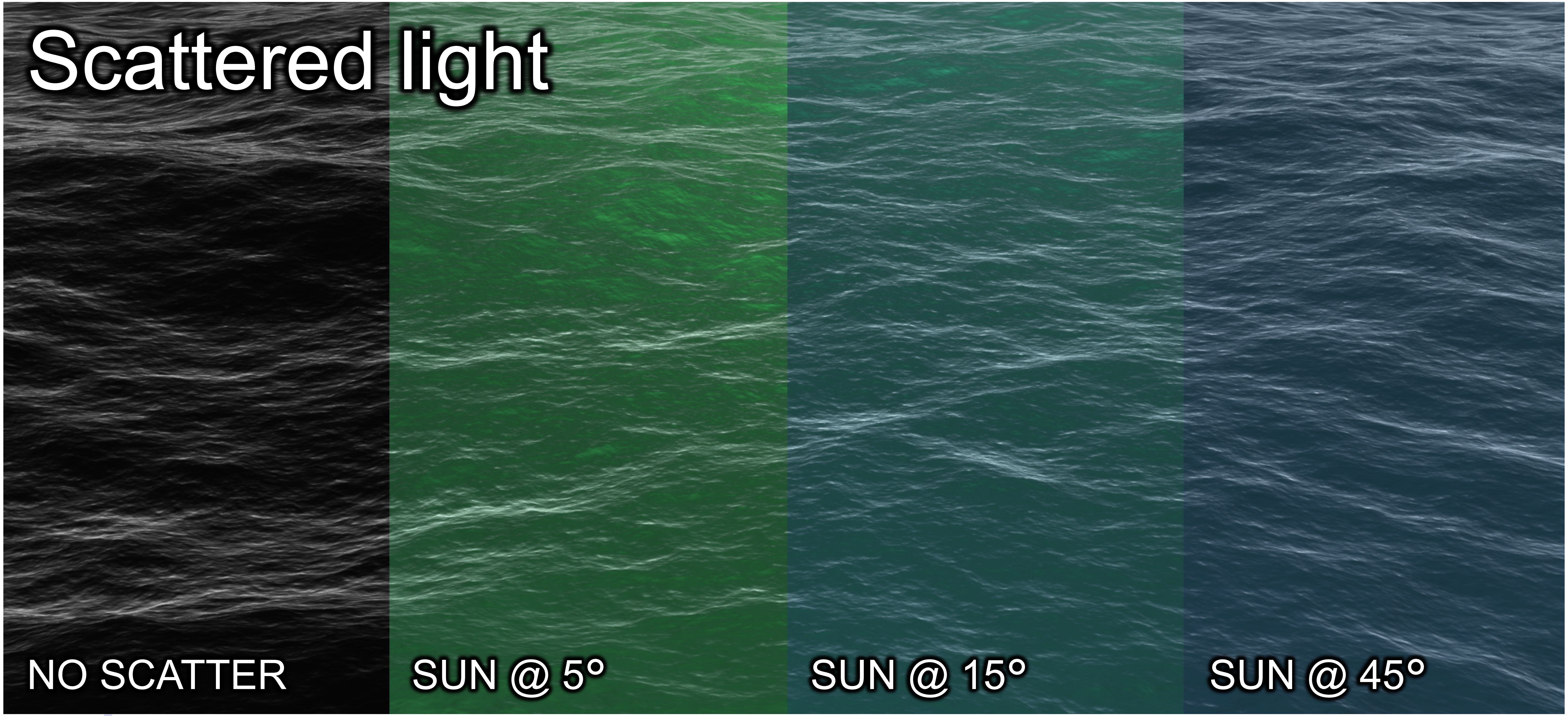
P_f – density of air bubbles spread in water

$\langle\omega_a, \omega_b\rangle$ – max(0, $(\omega_a \cdot \omega_b)$)

Scattered light



Scattered light



Putting it all together

Now combining it all together:

$$\boldsymbol{L_{to\_eye} = (1 - F)L_{scatter} + \int_{\Omega} L_{sun} + F \int_{\Omega} L_{env}}$$

Note $\boldsymbol{F}$ is already taken in account in analytical $\int \boldsymbol{L_{sun}}$

Adding surface foam:

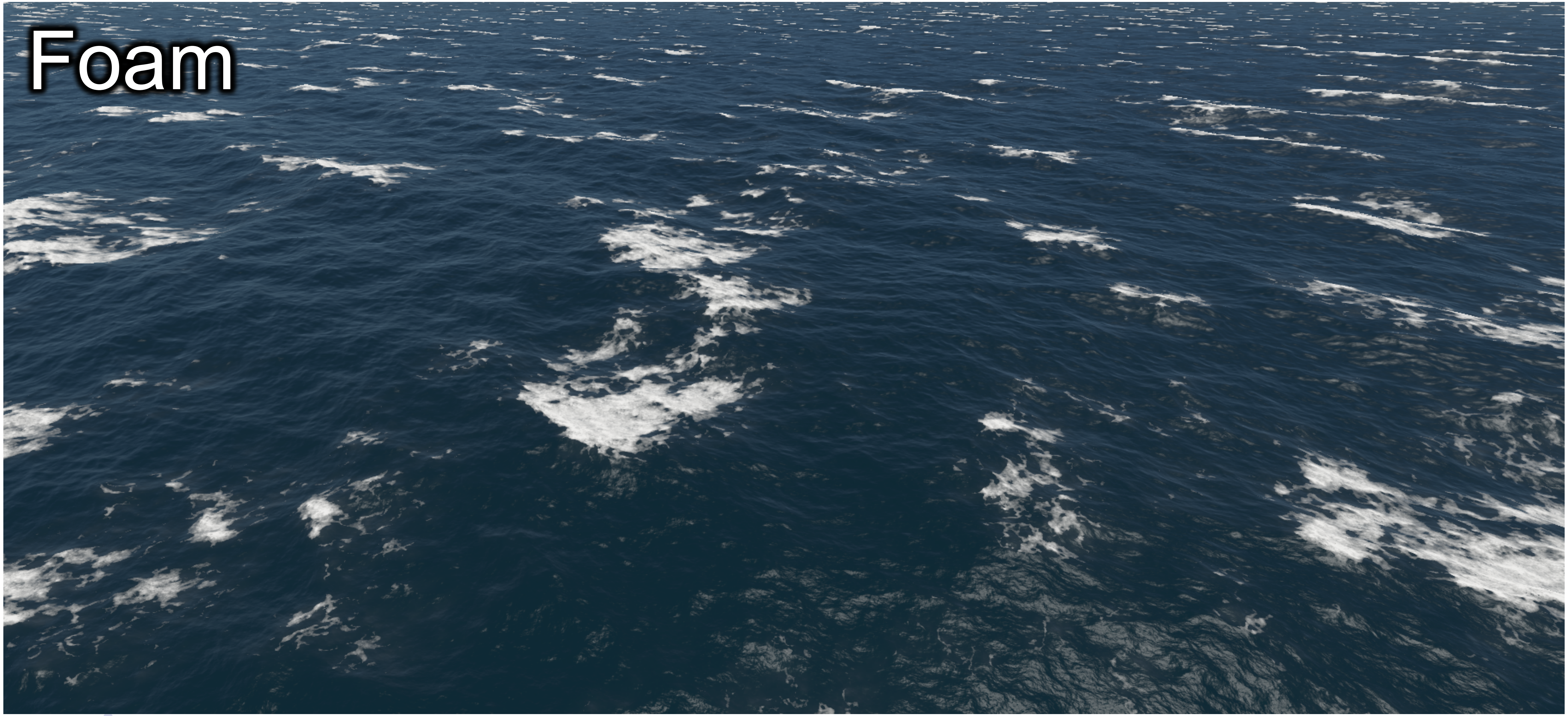
- Calculate foam color
- *Lerp* between foam color and $\boldsymbol{L_{to\_eye}}$ based on foam density
- Increase roughness in areas covered with foam for $\int \boldsymbol{L_{sun}}$

Foam

Calculate Jacobian of displacements per frequency band

- If above threshold
  - We are on a wave top
  - Inject some amount of “turbulent energy”
- Dissipate it over time (blur + fade)
- Sum up “turbulent energy” from all bands on rendering
  - Modulate foam texture by “turbulent energy”

Foam



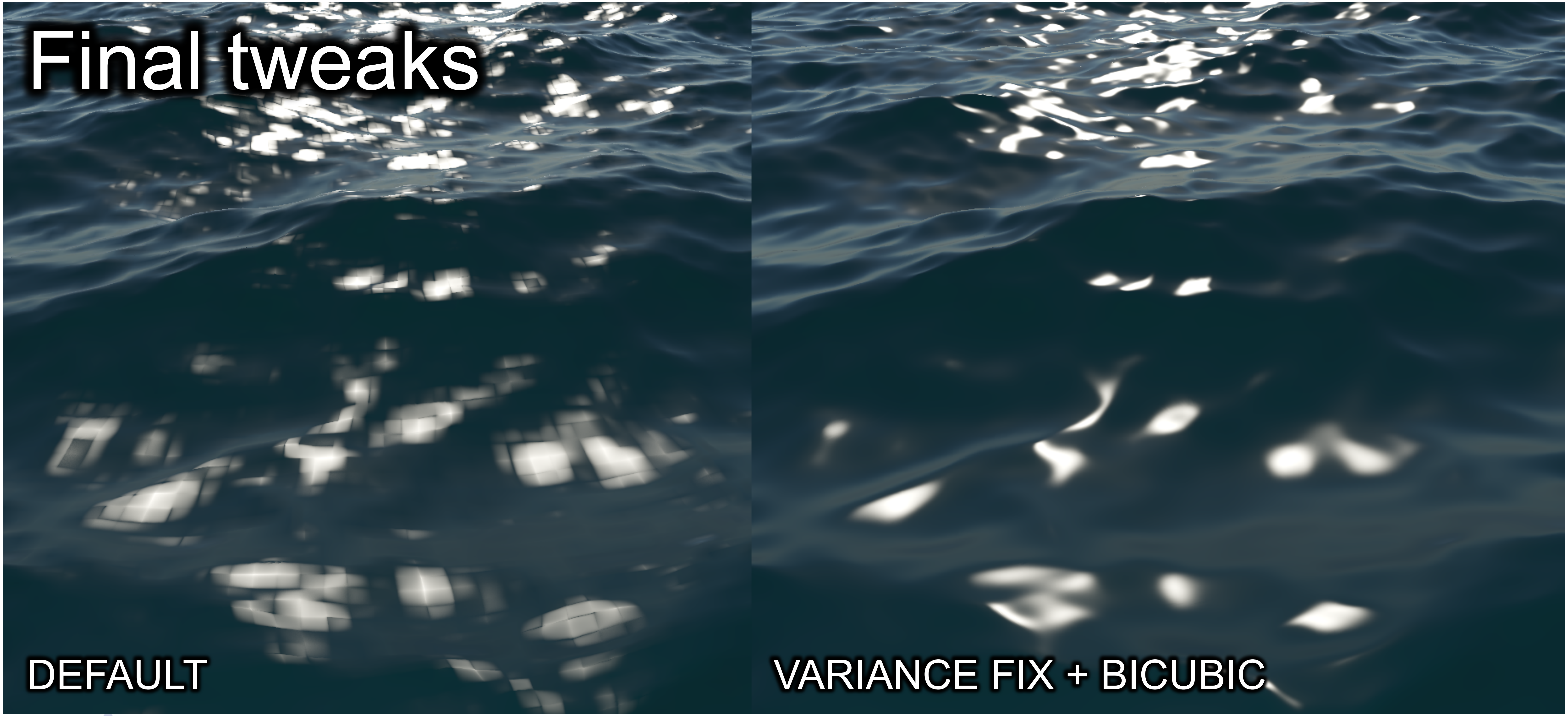
Final tweaks

Bilinear magnification is not a linear operator: $\delta_x^2 \neq E\left[x_{\bar{n}}^2\right] - E^2[x_{\bar{n}}]$

To fix this, we calculate mip using ddx and ddy, and:

- $Lerp$ second order moments to squares of first order moments: $E_{new}\left[x_{\bar{n}}^2\right] = lerp\left(E^2[x_{\bar{n}}], E\left[x_{\bar{n}}^2\right], clamp(0.25mip, 0, 1)\right)$, same for $y_{\bar{n}}$, effectively lerping variance to 0
- $Lerp$ from bilinear to bicubic filtering for most detailed frequency band

Final tweaks

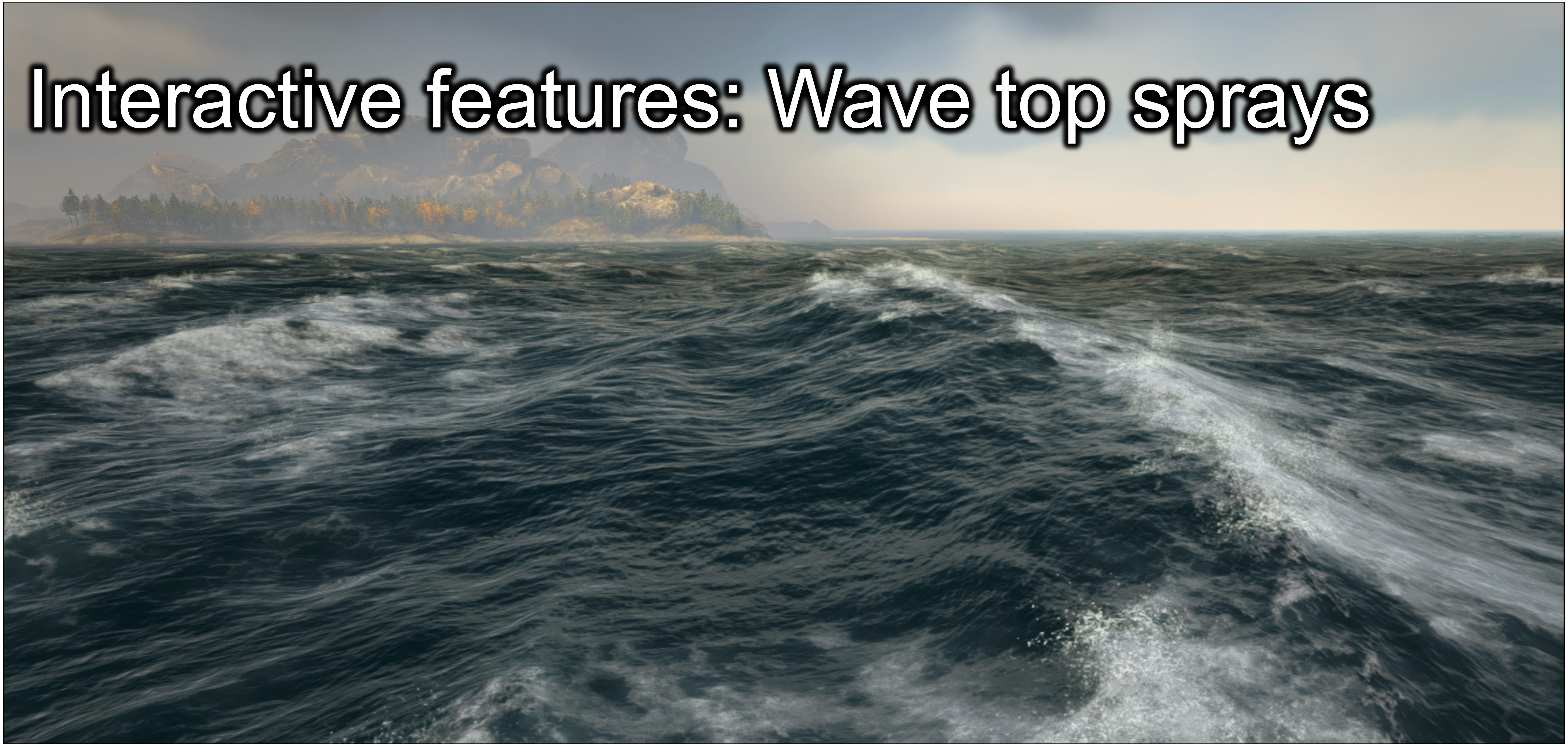


Final tweaks

Undersampling and crawling geometry:

- Fade displacements to zero at distance
- Bands start fading away at ~30 world space periods from camera

Interactive features: Wave top sprays



Wave top sprays

We want to spawn particles:

- Only in camera view
- Only where the waves crest and create whitecaps
- Should work within UE4's cascade particle system
  - No spawning from GPU
  - Simulation is done in a pixel shader with textures for Position/Time/Velocity

Wave top sprays

Solution:

- Custom emitter in camera frustum
  - Emit particles everywhere in view
  - Don’t simulate or render these yet
- Use particle location to sample world space foam/whitecap textures
- Allow GPU particle simulation to kill particles which are not on whitecaps
- Start actual simulation and rendering for valid particles

Wave top sprays

```
// Get vertex and surface attributes (same as for rendering)
VERTEX_OUTPUT Vert = GetDisplacedVertex(Particle.Pos)
SURFACE_PARAMETERS Surf = GetSurfaceParameters(Vert.Attributes);

// test if this particle is in a whitecap
if( (Surf.foam_wave_hats > Simulation.WaveHatThreshold)
{
   Particle.Pos += Vert.Displacement; // apply displacement
   Particle.LifeTime = 0.0f; // start simulation and allow rendering
} else {
   Particle.LifeTime = -1.0f; // particle doesn't render until lifetime > 0
   return; // skip rest of simulation
}
```

Wakes, explosions

Wakes, explosions

Explosions currently done with spray particles & sprites

Wakes currently done with foam sprites

Do not affect sea surface displacements

We can do better!

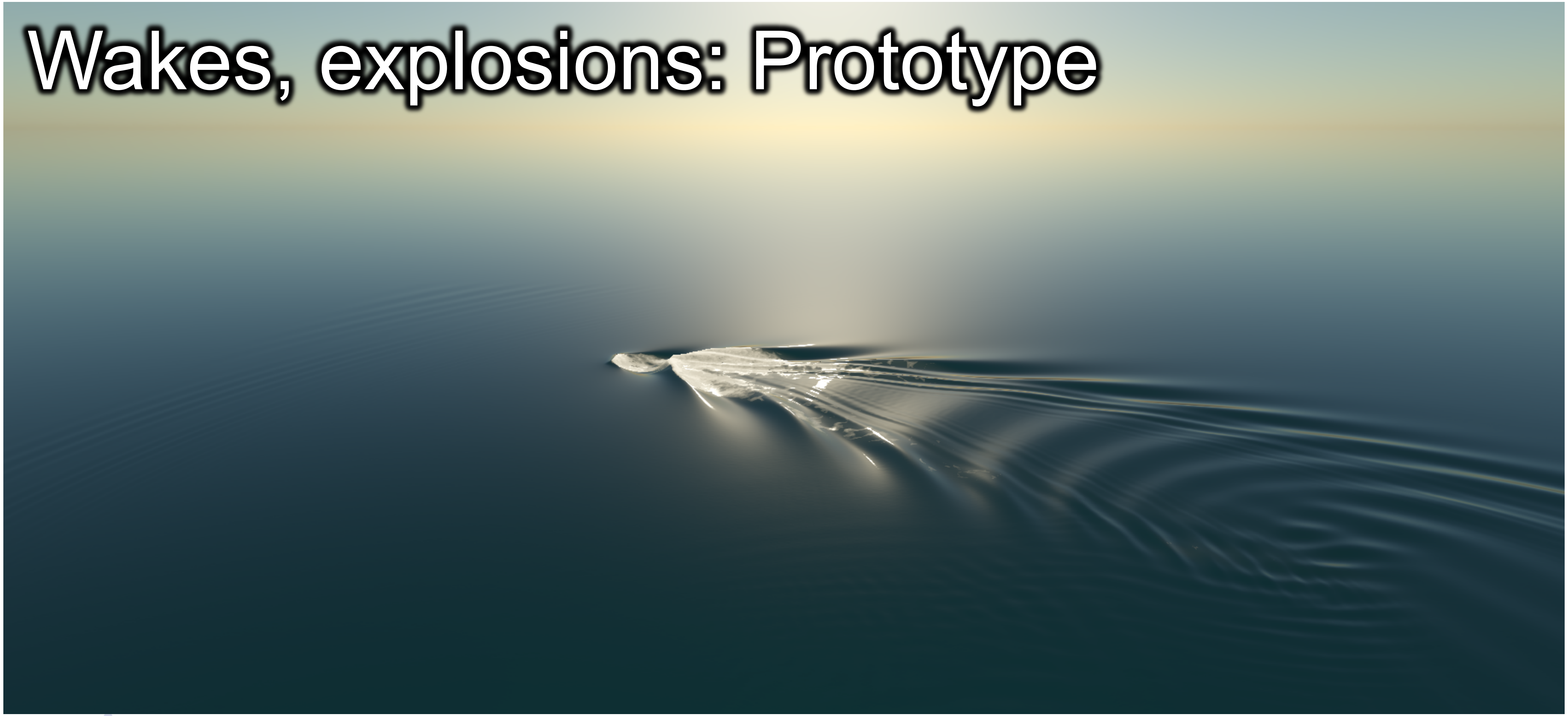
Wakes, explosions: Prototype

Simulate on a grid using Tessendorf's eWave solver:

Complex displacements and velocity potential per grid cell. On each simulation step:

- Inject displacements
- Convert to frequency domain
- Generate evolving operators V(dT) that respect dispersion relation
- Apply evolving operators
- Generate lateral displacements
- Convert back to spatial domain

Wakes, explosions: Prototype



Wakes, explosions: Prototype

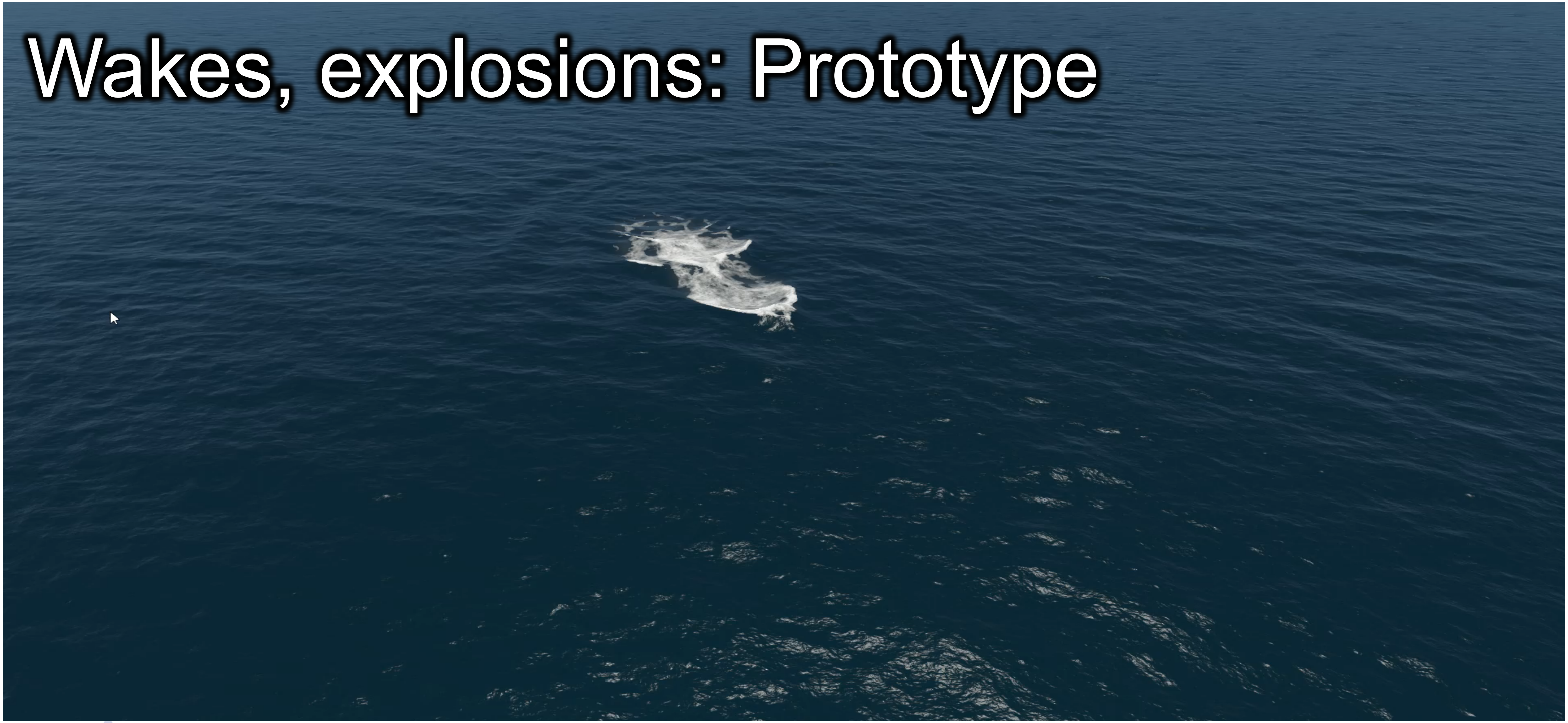
Very cool looking results with natural waves and ideal Kalvin wakes etc, but:

- Solution is periodic
  - We apply exponential dampening on the edges of simulation domain
- Does not simulate foam
  - We inject and evolve foam: same math as wind waves
  - We combine wind waves foam and interactive foam

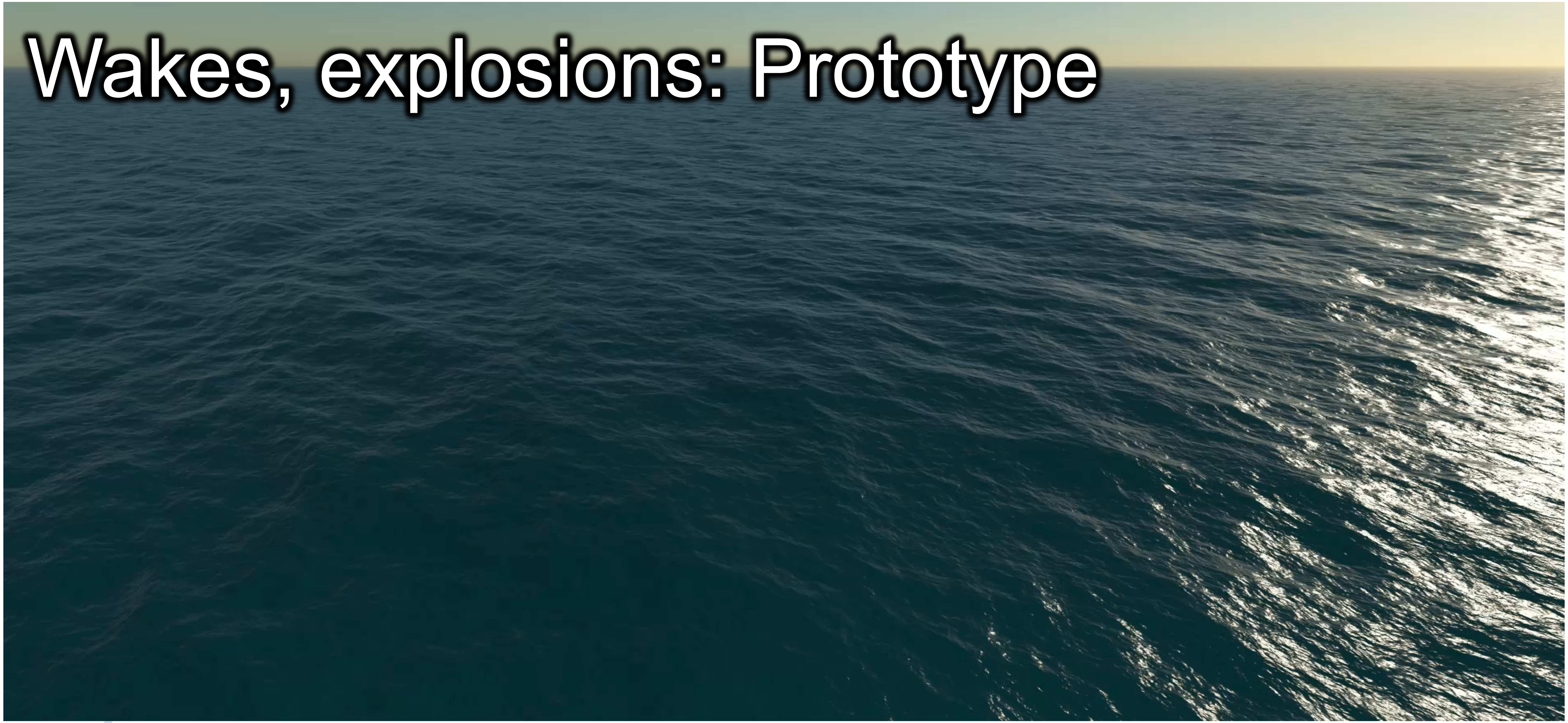
Overall: 2 forward FFTs, multiplication in frequency space, and 4 inverse FFTs

Wakes, explosions: Prototype

Wakes, explosions: Prototype



Wakes, explosions: Prototype



Wrapping up: timings

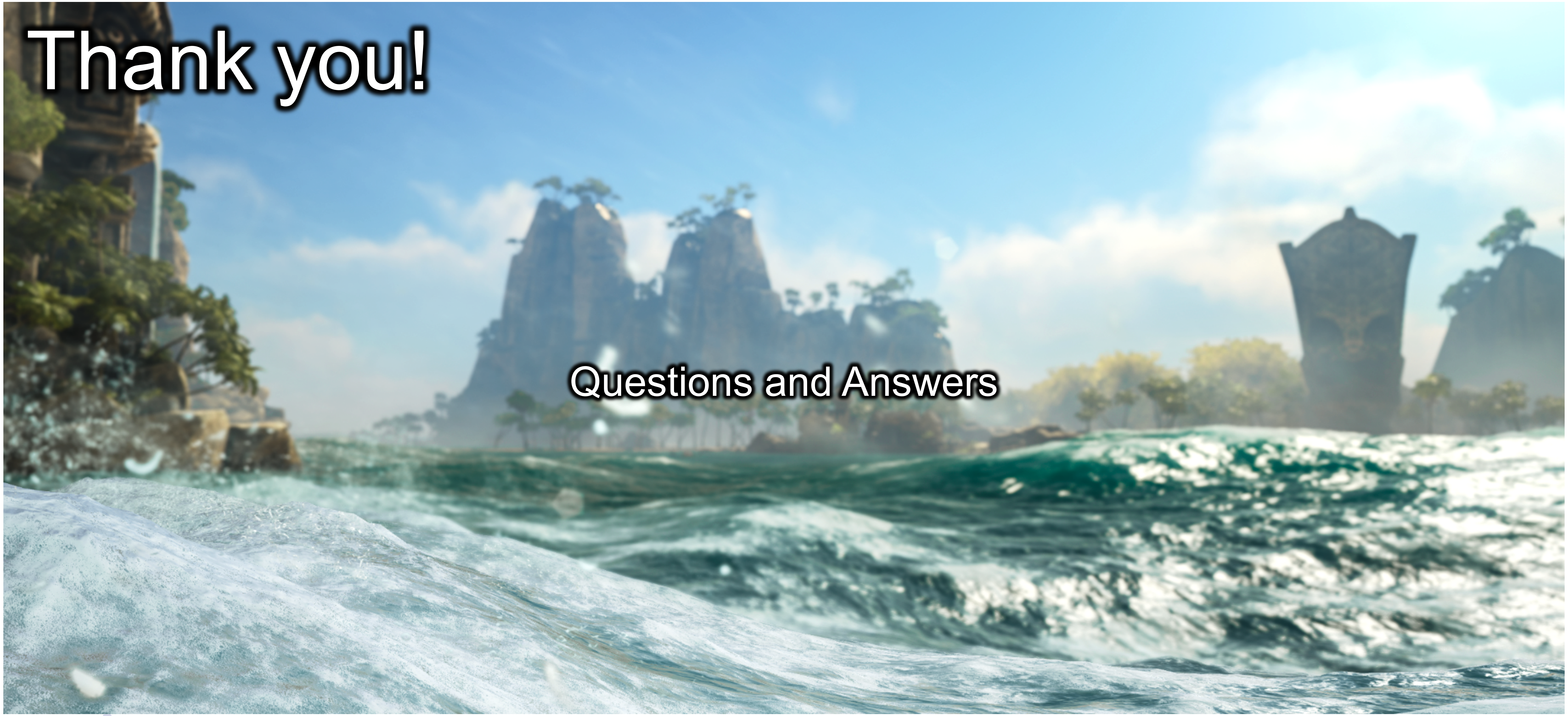
Wind waves simulation time on GPU (max quality):

AMD RADEON VII	NVIDIA GeForce RTX2080
0.5 msec	0.5 msec

Interactive waves simulation time on GPU (normal quality):

AMD RADEON VII	NVIDIA GeForce RTX2080
0.8 msec	0.6 msec

Thank you!



Questions and Answers