EDAF80 Introduction to Computer Graphics

Seminar 4

Water shader

Pierre Moreau

2019 Slides by Carl Johan Gribel, 2010-13 and Michael Dogget, 2014-17

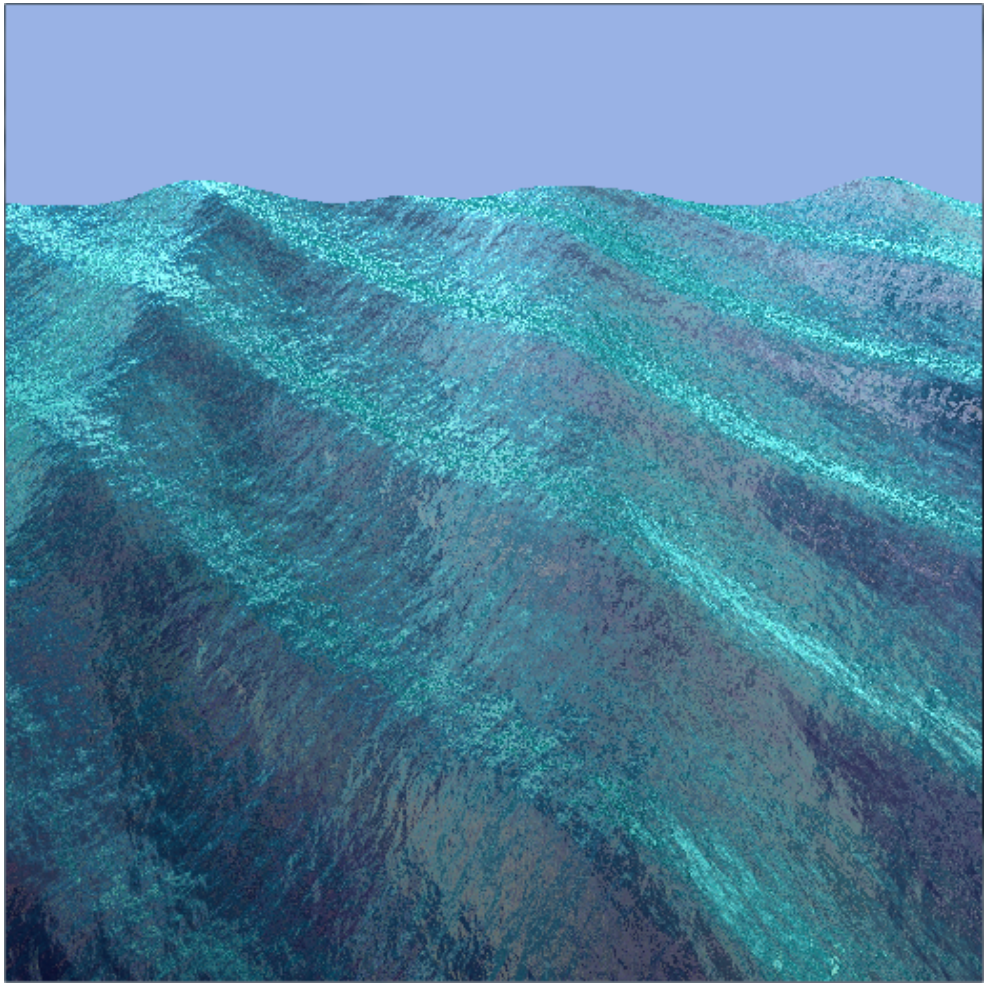
Preparations

- Finch, M. "Effective Water Simulation from Physical Models"
  Excerpt from *GPU Gems* book

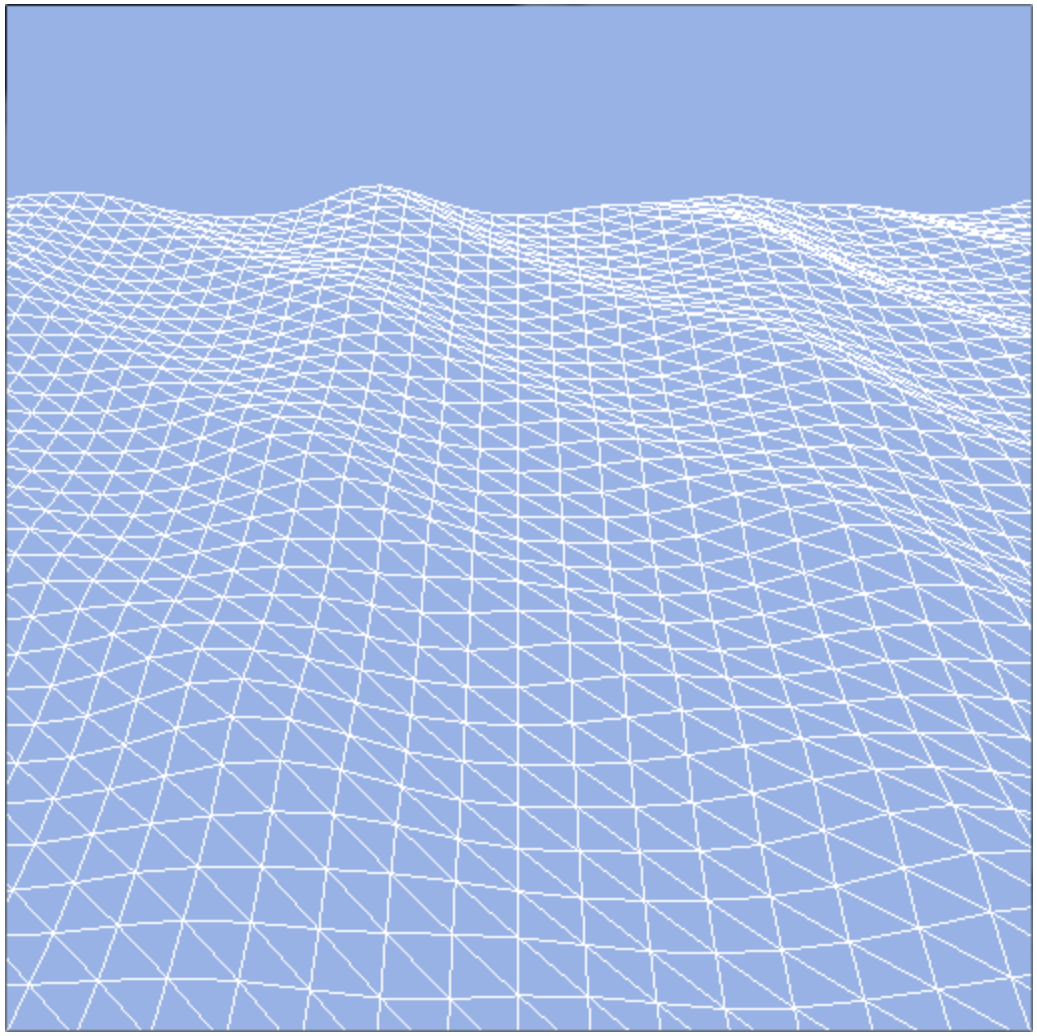
- **Note**: the article uses notation (**B**, **T**, **N**) ~~(**t**, **b**, **n**)~~

Today

- Water shader
  Waves
  Shallow & deep color
  Fresnel reflection
  Fresnel refraction
  Animated normal mapping



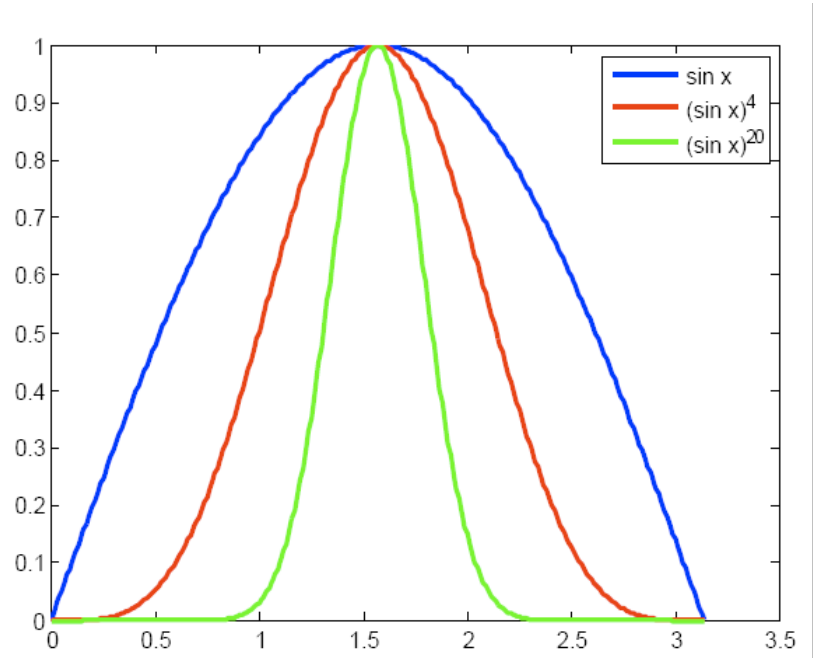
Waves



Waves

- Sum of sines
- Form sharper crest by raising to an exponent *k* (**sharpness**)

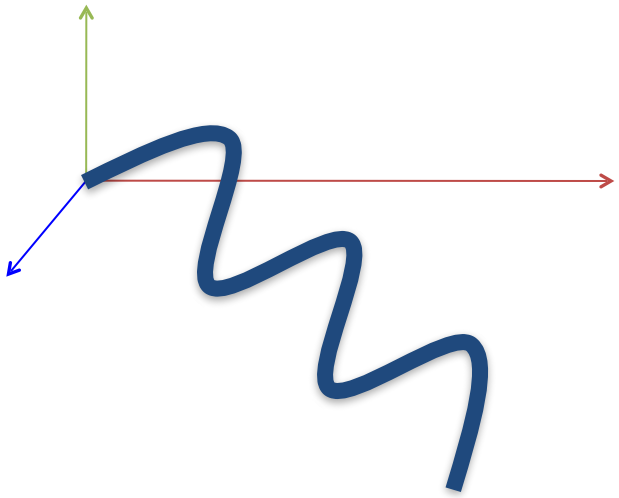
$$\sin(x) \rightarrow \sin(x)^k$$



- Similar to **shininess** in the phong model

One wave

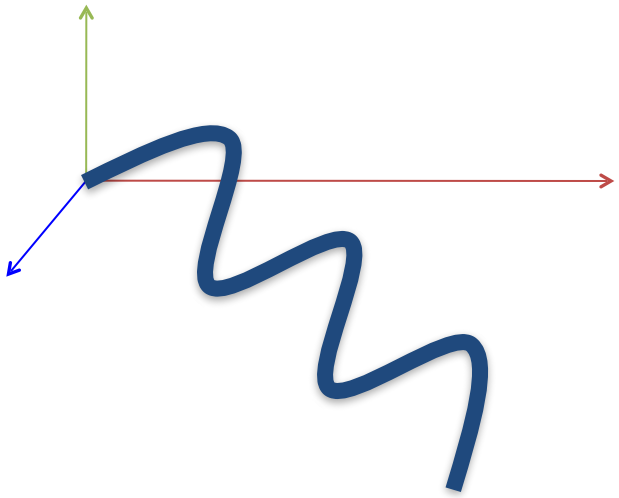
A = amplitude
D = (D_x, D_z) = direction of travel
f = frequency
p = phase
k = sharpness
t = time
(x, z) = position on plane of water surface



$$y = G(x, z, t) = A(\sin((D_x \cdot x + D_z \cdot z) \cdot f + tp) \cdot 0.5 + 0.5)^k$$

One wave

A = amplitude
D = (D_x, D_z) = direction of travel
f = frequency
p = phase
k = sharpness
t = time
(x, z) = position on plane of water surface



$$y = G(x, z, t) = A(\sin((D_x \cdot x + D_z \cdot z) \cdot f + tp) \cdot 0.5 + 0.5)^k$$

$$\frac{\partial G}{\partial x} = 0.5kfA(\sin((D_x \cdot x + D_z \cdot z) \cdot f + tp) \cdot 0.5 + 0.5)^{k-1} \cdot \cos((D_x \cdot x + D_z \cdot z) \cdot f + tp) \cdot D_x$$

$$\frac{\partial G}{\partial z} = 0.5kfA(\sin((D_x \cdot x + D_z \cdot z) \cdot f + tp) \cdot 0.5 + 0.5)^{k-1} \cdot \cos((D_x \cdot x + D_z \cdot z) \cdot f + tp) \cdot D_z$$

Sum of waves

One wave

$$y = G(x, z, t) = A(\sin((D_x \cdot x + D_z \cdot z) \cdot f + tp) \cdot 0.5 + 0.5)^k$$

Sum of waves

$$H(x, z, t) = \sum G_i$$

$$\frac{\partial H}{\partial x} = \sum \frac{\partial G_i}{\partial x}$$

$$\frac{\partial H}{\partial z} = \sum \frac{\partial G_i}{\partial z}$$

Implementing a function in GLSL

- Definition and declaration work the same way as in C or C++.
  See https://www.khronos.org/opengl/wiki/Core_Language_(GLSL)#Functions for more details

- The types of arguments can be prefixed by a keyword:

  - `in`: the argument is an input (default if nothing is specified)

  - `out`: the argument is an output

  - `inout`: the argument is an input and an output

- For example:

```
void f1(in int s) { s = 3; }
void f2(out int s) {
    // the value of s in undefined before the assignment below
    s = 3;
}
void f3(inout int s) {
    // s is defined
    s = 4;
}

int s = 1;
f1(s); // s is still 1 after the call
f2(s); // s is now 3
f3(s); // s is now 4
```

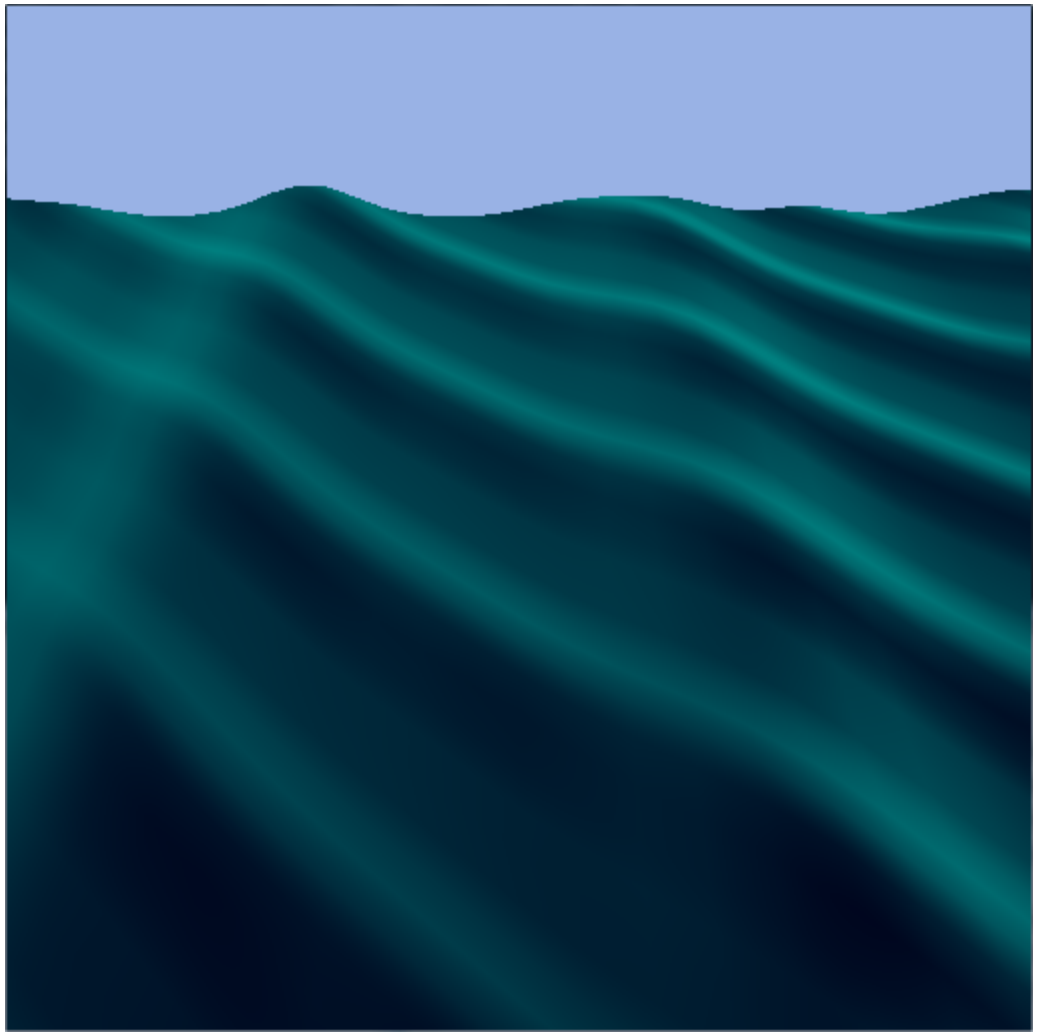
Implementing a function in GLSL

```
layout (location = 0) in vec3 vertex;
layout (location = 1) in vec3 normal;

uniform mat4 vertex_model_to_world;
uniform mat4 normal_model_to_world;
uniform mat4 vertex_world_to_clip;

out VS_OUT {
    vec3 vertex;
    vec3 normal;
} vs_out;

float wave(vec2 position, vec2 direction, float amplitude, float frequency,
           float phase, float sharpness, float time)
{
    return amplitude * pow(sin((position.x * direction.x + position.y * direction.y)
                               * frequency + time * phase) * 0.5 + 0.5, sharpness);
}

void main()
{
    vec3 displaced_vertex = vertex;
    displaced_vertex.y += wave(vertex.xz, vec2(-1.0, 0.0), /* … fill in */);

    vs_out.vertex = vec3(vertex_model_to_world * vec4(displaced_vertex, 1.0));
    vs_out.normal = vec3(normal_model_to_world * vec4(normal, 0.0));

    gl_Position = vertex_world_to_clip * vertex_model_to_world * vec4(vertex, 1.0);
}
```

Implementing a function in GLSL

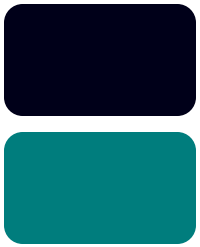
```
layout (location = 0) in vec3 vertex;
layout (location = 1) in vec3 normal;

uniform mat4 vertex_model_to_world;
uniform mat4 normal_model_to_world;
uniform mat4 vertex_world_to_clip;

out VS_OUT {
    vec3 vertex;
    vec3 normal;
} vs_out;

float wave(vec2 position, vec2 direction, float amplitude, float frequency,
           float phase, float sharpness, float time)
{
    return amplitude * pow(sin((position.x * direction.x + position.y * direction.y)
                               * frequency + time * phase) * 0.5 + 0.5, sharpness);
}

void main()
{
    vec3 displaced_vertex = vertex;
    displaced_vertex.y += wave(vertex.xz, vec2(-1.0, 0.0), /* … fill in */);

    vs_out.vertex = vec3(vertex_model_to_world * vec4(displaced_vertex, 1.0));
    vs_out.normal = vec3(normal_model_to_world * vec4(normal, 0.0));

    gl_Position = vertex_world_to_clip * vertex_model_to_world * vec4(vertex, 1.0);
}
```

Water color



Water color

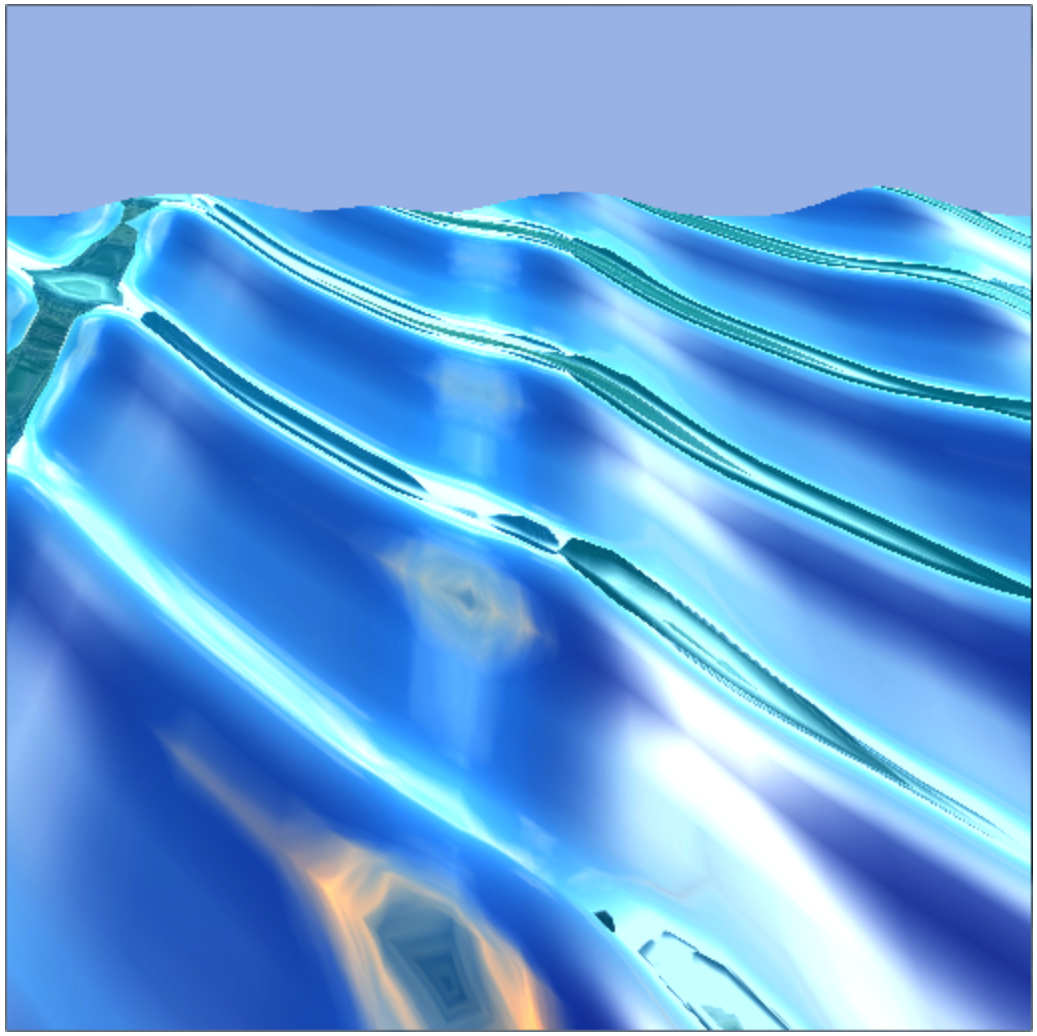
- $\text{Color}_{\text{deep}}$ = { 0.0, 0.0, 0.1, 1.0 }
- $\text{Color}_{\text{shallow}}$ = { 0.0, 0.5, 0.5, 1.0 }



- $\mathbf{n} = (-\partial H/\partial x, 1, -\partial H/\partial z)$
- *facing* = 1 – max($\mathbf{V} \cdot \mathbf{n}$, 0)

- $\text{Color}_{\text{water}}$ = mix($\text{Color}_{\text{deep}}$, $\text{Color}_{\text{shallow}}$, *facing*)

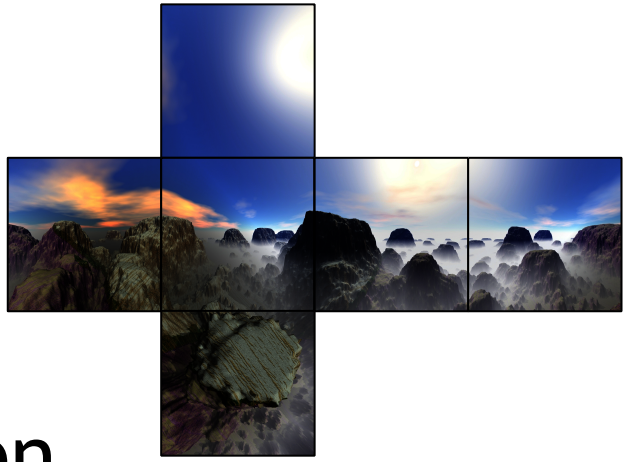
Reflection



Reflection

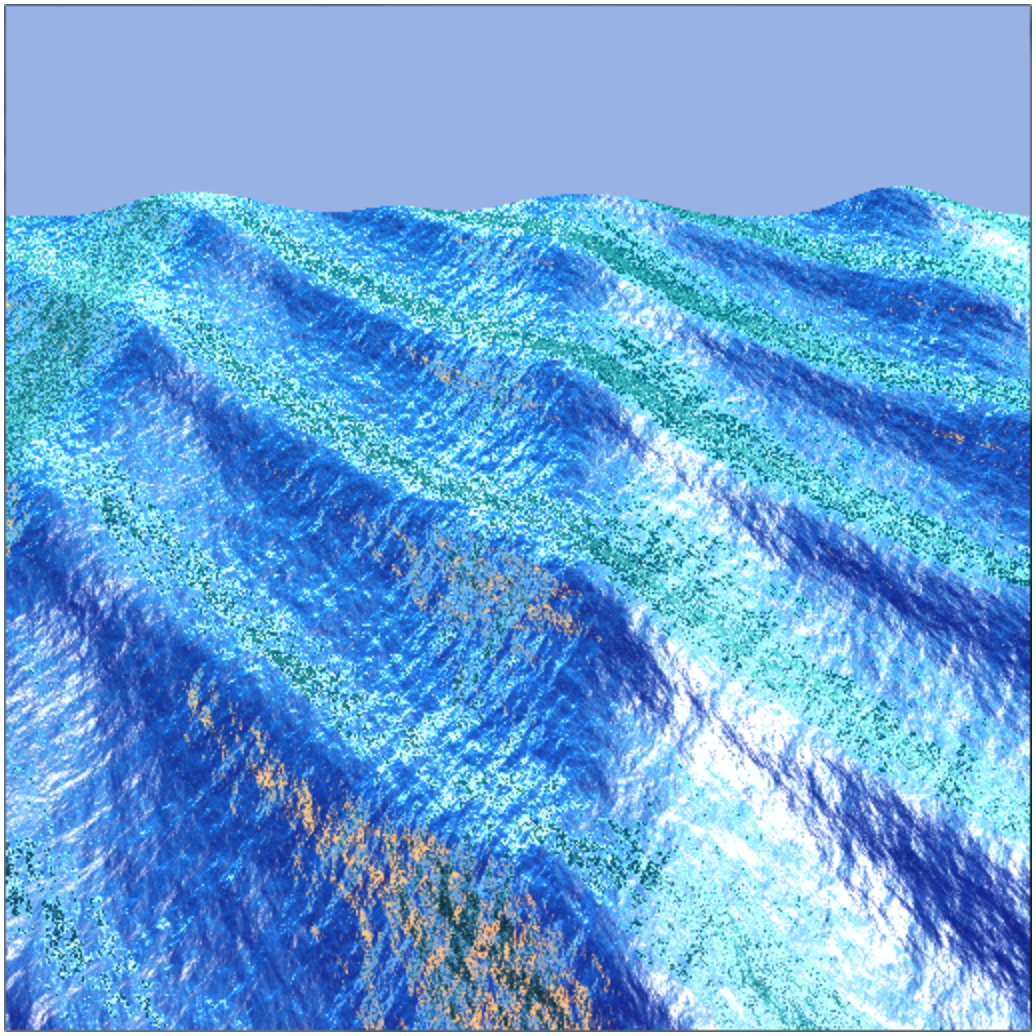
= Cube mapping, as in assignment 3

- $\mathbf{n} = (-\partial H/\partial x, 1, -\partial H/\partial z)$
  $\mathbf{R} = \text{reflect}(-\mathbf{V}, \mathbf{n})$
- *cloudy hills* cubemap set



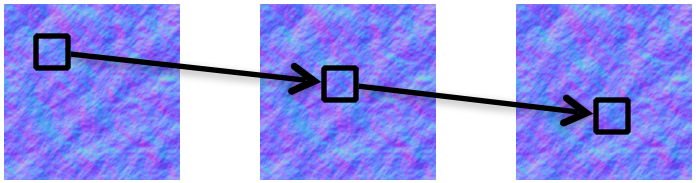
- $\text{Color} = \text{Color}_{\text{water}} + \text{reflection}$

Animated Normal mapping



Animated Normal mapping

- ”Sliding windows” – superposition from multiple, time-dependent bump map coord’s



- Vertex shader: calculate coordinate pairs
  Fragment shader: read and superposition

Animated Normal mapping: coordinates

texScale = (8, 4)

normalTime = mod(time, 100.0)

normalSpeed = (-0.05, 0)

```
normalCoord0.xy =
      TexCoord.xy*texScale + normalTime*normalSpeed
normalCoord1.xy =
      TexCoord.xy*texScale*2 + normalTime*normalSpeed*4
normalCoord2.xy =
      TexCoord.xy*texScale*4 + normalTime*normalSpeed*8
```

Animated Normal mapping: read, remap and superposition

$\mathbf{n}_i$ = texure(NormalTex, normalCoord_i) * 2 – 1

$\mathbf{n}_{\text{bump}} = \text{normalize}(\sum \mathbf{n}_i)$ (*tangent space*)

Tangent space

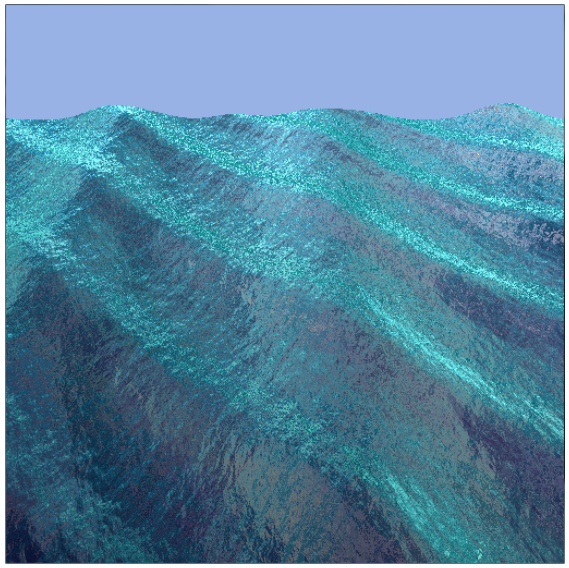
- Tangent space
  t = (1, ∂H/dx, 0)
  b = (0, ∂H/dz, 1)
  n = (-∂H/∂x, 1, -∂H/∂z)

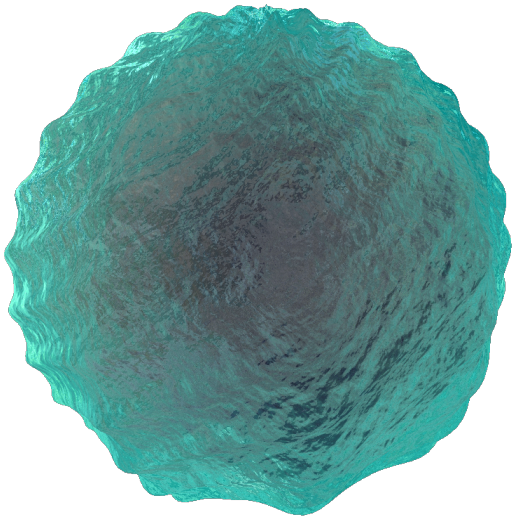
- tangent space -> world space
  TBN * $\mathbf{n}_{bump}$

More complex surfaces

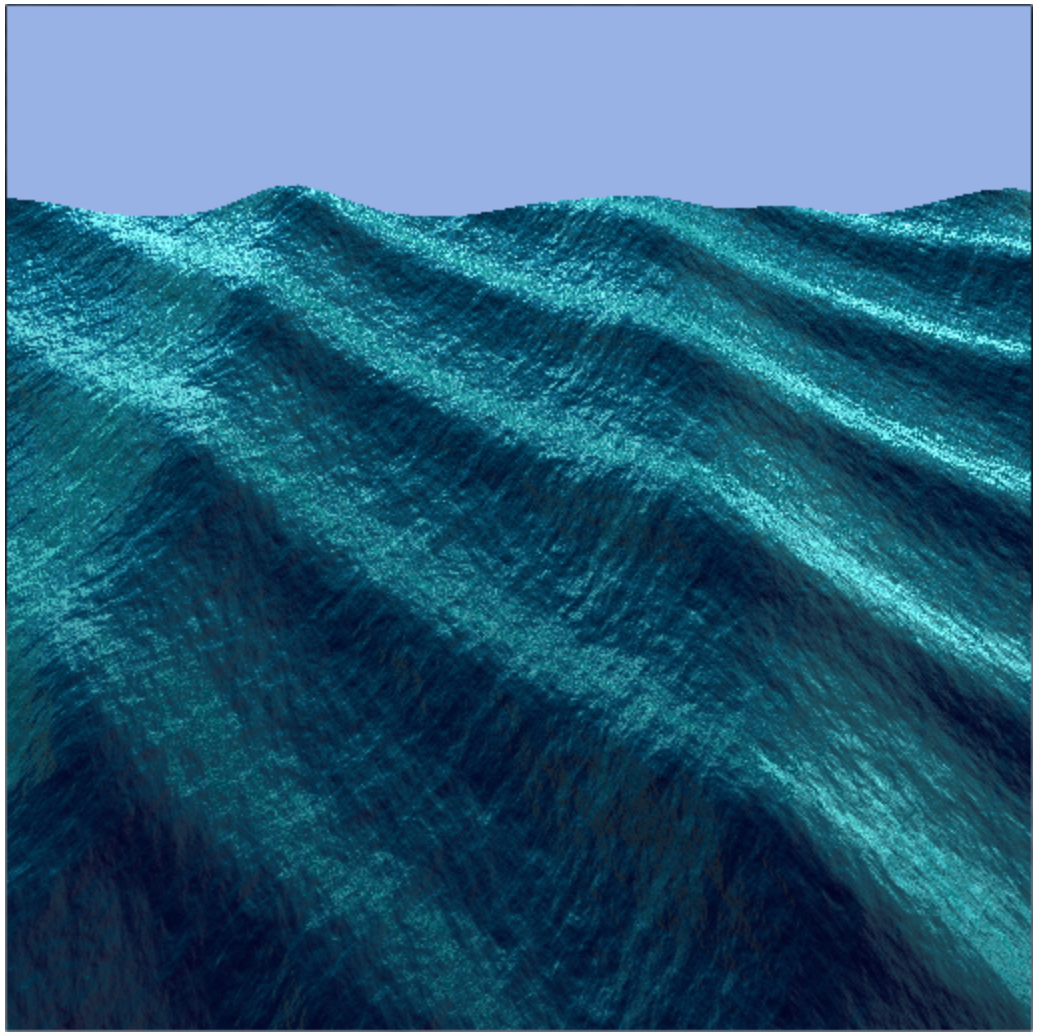
- In general, *wave* $\mapsto$ *surface* $\mapsto$ *model* $\mapsto$ *world*:

  World * $TBN_{surface}$ * TBN_{water} * n_{bump}





Fresnel reflection



Fresnel terms

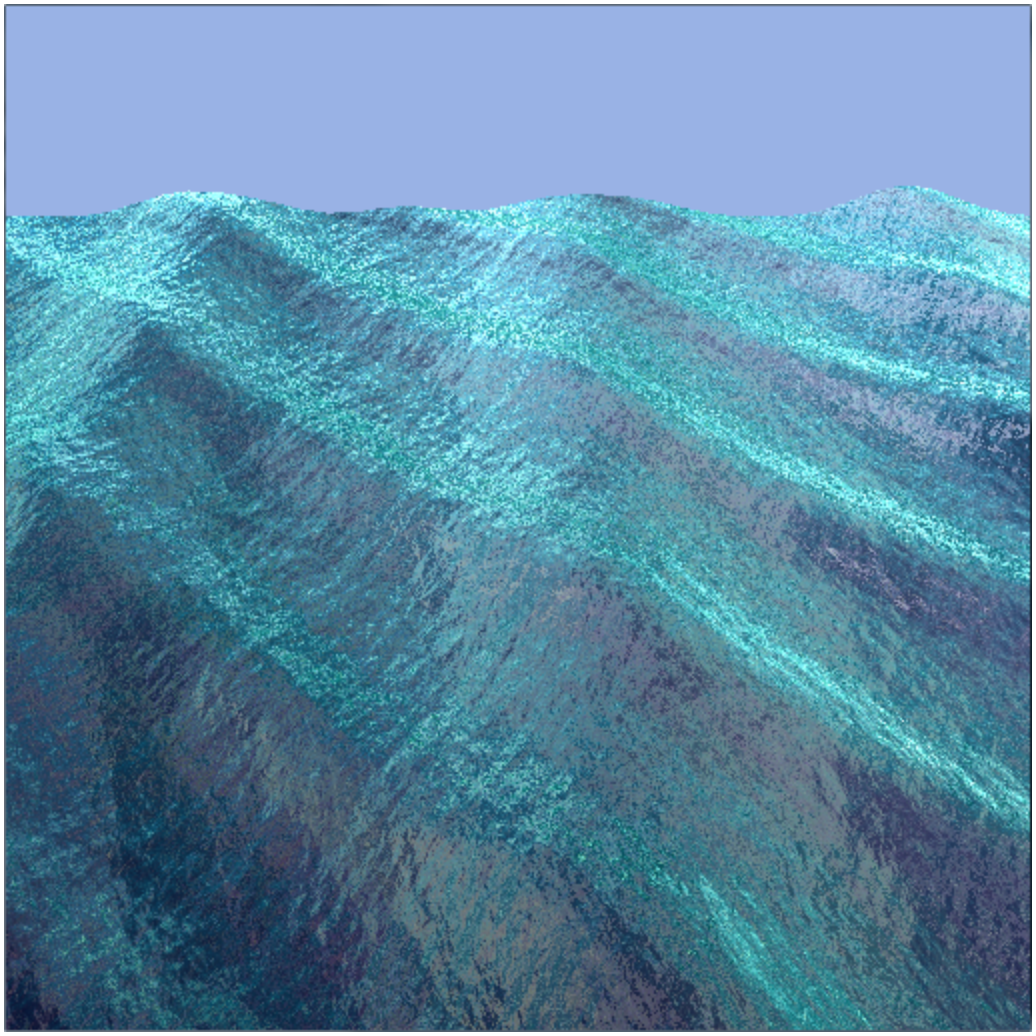
- How much light reflects at a glancing angle
  And ~ how much refracts

fastFresnel = $R_0 + (1 - R_0)*(1 - \mathbf{V}\cdot\mathbf{n})^5$

$R_0 = ((n_1-n_2)/(n_1+n_2))^2$
Air to water: $R_0 = 0.02037$

- Color = $\text{Color}_{\text{water}}$ + reflection * fresnel

Refraction



Refraction

- Use function refract(I, N, n);
- takes refraction index *n* as parameter
  n = 1.33 for *air -> water*)
  n = 1.0/1.33 for *water -> air*)

- Color = $Color_{water}$ +
  reflection * fresnel +
  refraction * (1 - fresnel)
  - reflection and refraction are looked up in the cube map

Final result

Assignment 4

- **Water shader**
  Use existing framework code Assignment 4 target (**`src/EDAF80/assignment4.cpp`**)
  - Use createQuad
  - Increase tessellation to ~ 50x50
  - Add texture coordinates
  - Start with a basic shader, diffuse.vert/.frag

Assignment 4

- **Resources**: *waves.png* bump map
  cloudy hills cubemap set
  included in res directory