Water Shader

EDAF80: Computer Graphics

Rikard Olajos

AGENDA

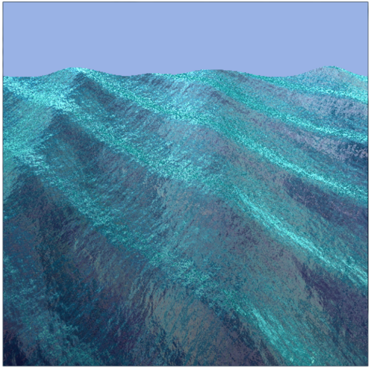
1. Introduction
2. Waves
3. Shading
4. Assignment 4

ARTICLE

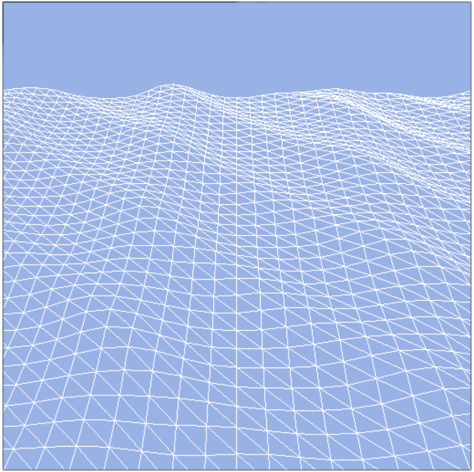
- Finch, M.: Effective Water Simulation from Physical Models
- Excerpt from ***GPU Gems***
- Note: uses $(\mathbf{B}, \mathbf{T}, \mathbf{N})$ instead of $(\mathbf{t}, \mathbf{b}, \mathbf{n})$
- 2nd note: there is a mix-up of terminology (phase/frequency/wave number)

OVERVIEW

- Waves
- Shallow & deep colour
- Fresnel reflection
- Fresnel refraction
- Animated normal mapping



WAVES

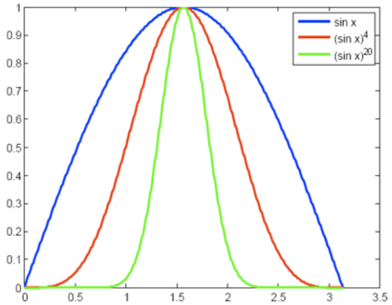


WAVES

- 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
- φ = phase constant
- k = sharpness
- t = time
- (x, z) = position on plane

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

ONE WAVE

- A = amplitude
- $D = (D_x, D_z)$ = direction of travel
- f = frequency
- φ = phase constant
- k = sharpness
- t = time
- (x, z) = position on plane

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

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

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

SUM OF WAVES

- One wave

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

- Sum of waves

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

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

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

GLSL FUNCTIONS

- Parameter qualifiers
  - `in`: copied into the function (default)
  - `out`: copied out of the function
  - `inout`: copied into and out of the function

```
void my_function(in float a, out float b, inout float c) {
    b = a + c;
    c = 7.7;
}

float a = 2.2; float b = 3.3; float c = 4.4;
my_function(a, b, c);   // b = 6.6, c = 7.7
```

- Can not be recursive
- More info in the OpenGL wiki

GLSL WAVE FUNCTION

```
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;
/* Add time uniform */

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 + phase * time) * 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);
}
```

ADDING TIME

- Keep track of elapsed time

```
float elapsed_time_s = 0.0f;
```

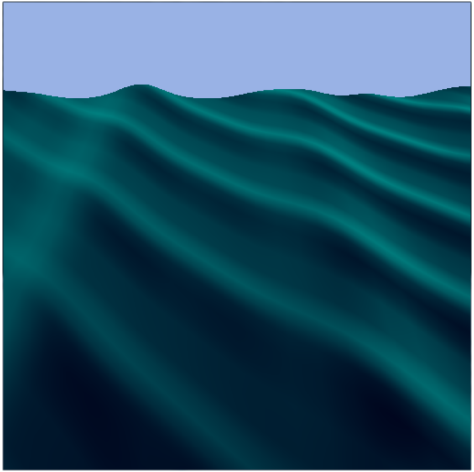
- Update each frame in render loop

```
elapsed_time_s += std::chrono::duration<float>(deltaTimeUs).count();
```

- Send time to shader program

```
glUniform1f(glGetUniformLocation(program, "t"), elapsed_time_s);
```

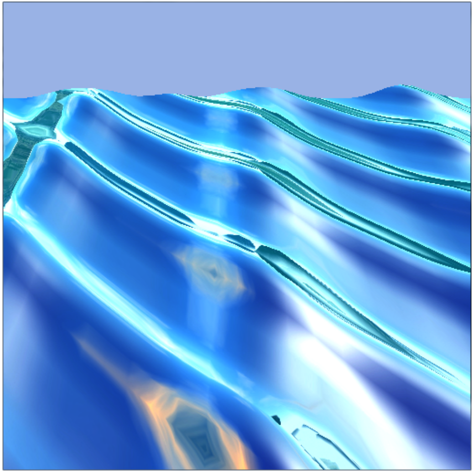
WATER COLOUR



WATER COLOUR

- $\texttt{color}_{\texttt{deep}} = (0.0, 0.0, 0.1, 1.0)$
- $\texttt{color}_{\texttt{shallow}} = (0.0, 0.5, 0.5, 1.0)$
- $\mathbf{n} = (-\partial H/\partial x, 1, -\partial H/\partial z)$
- $\texttt{facing} = 1 - \max(\mathbf{V} \cdot \mathbf{n}, 0)$
- $\texttt{color}_{\texttt{water}} = \mathrm{mix}(\texttt{color}_{\texttt{deep}}, \texttt{color}_{\texttt{shallow}}, \texttt{facing})$

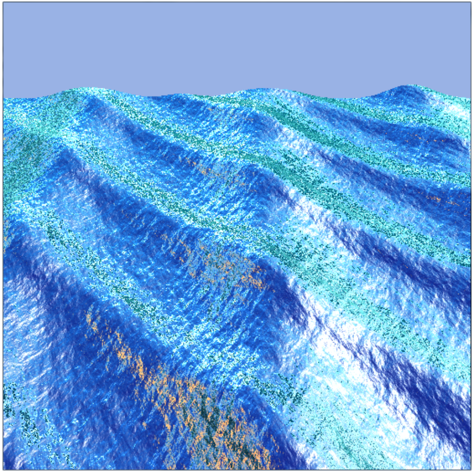
REFLECTION



REFLECTION

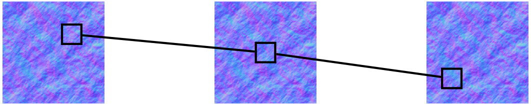
- Cube mapping, as in assignment 3
- $\mathbf{n} = (-\partial H/\partial x, 1, -\partial H/\partial z)$
- $\mathbf{R} = \text{reflect}(-\mathbf{V}, \mathbf{n})$
- $\texttt{color} = \texttt{color}_{\texttt{water}} + \texttt{reflection}$
- Reflection is looked up in the cube map

ANIMATED NORMAL MAPPING



ANIMATED NORMAL MAPPING

- "Sliding windows" — superposition from multiple, time-dependent normal map coordinates



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

ANIMATED NORMAL MAPPING: COORDINATES

```
vec2 texScale = vec2(8, 4);
float normalTime = mod(time, 100.0);
vec2 normalSpeed = vec2(-0.05, 0.0);

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;
```

ANIMATE NORMAL MAPPING: READ, REMAP, SUPERPOSITION

- $\mathbf{n}_i = \text{texture}(\texttt{normalTexture}, \texttt{normalCoord}_i) * 2 - 1$
- $\mathbf{n}_{\texttt{bump}} = \text{normalize}(\sum \mathbf{n}_i)$ (this is in *tangent space*)

TANGENT SPACE

- Equation for 3-D point

$$\mathbf{P}(x, z, t) = (x, H(x, z, t), z)$$

- Tangent space

$$\mathbf{t} = \left(\frac{\partial x}{\partial x}, \frac{\partial H}{\partial x}, \frac{\partial z}{\partial x}\right) = \left(1, \frac{\partial H}{\partial x}, 0\right)$$

$$\mathbf{b} = \left(\frac{\partial x}{\partial z}, \frac{\partial H}{\partial z}, \frac{\partial z}{\partial z}\right) = \left(0, \frac{\partial H}{\partial z}, 1\right)$$

$$\mathbf{n} = \mathbf{t} \times \mathbf{b} = \left(-\frac{\partial H}{\partial x}, 1, -\frac{\partial H}{\partial z}\right)$$

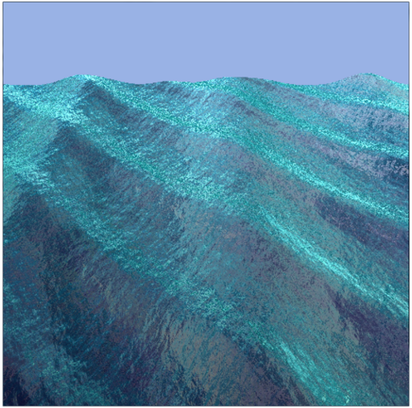
- tangent space $\rightarrow$ world space

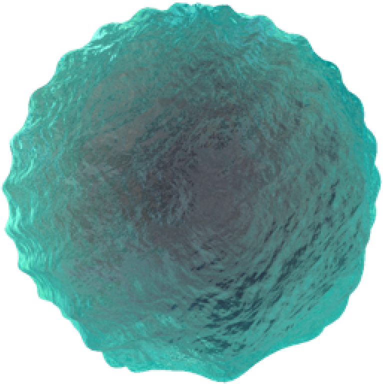
$$\mathbf{TBN} * \mathbf{n}$$

MORE COMPLEX SURFACES

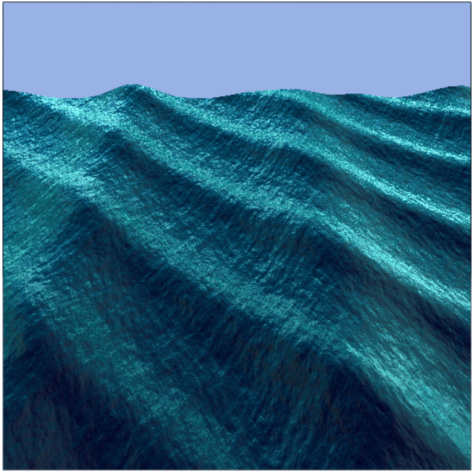
- In general: wave $\rightarrow$ surface $\rightarrow$ model $\rightarrow$ world

$$\mathbf{WORLD} * \mathbf{TBN}_{surface} * \mathbf{TBN}_{water} * \mathbf{n}$$





FRESNEL REFLECTION



FRESNEL TERMS

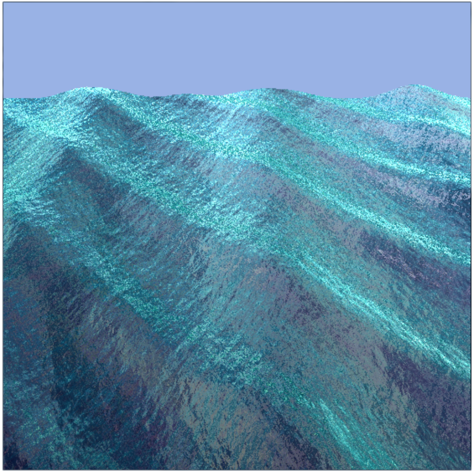
- How much light reflects at a glancing angle
- (and how much refracts)

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

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

- Air to water: $R_0 = 0.02037$
- $\texttt{color} = \texttt{color}_{\texttt{water}} + \texttt{reflection} * \texttt{fresnel}$

FRESNEL REFRACTION



FRESNEL REFRACTION

- Use function `refract(I, N, eta)`
- Takes *incident* vector, `I`, normal `N`, and ratio of refraction indices, `eta`
- Air $\rightarrow$ water

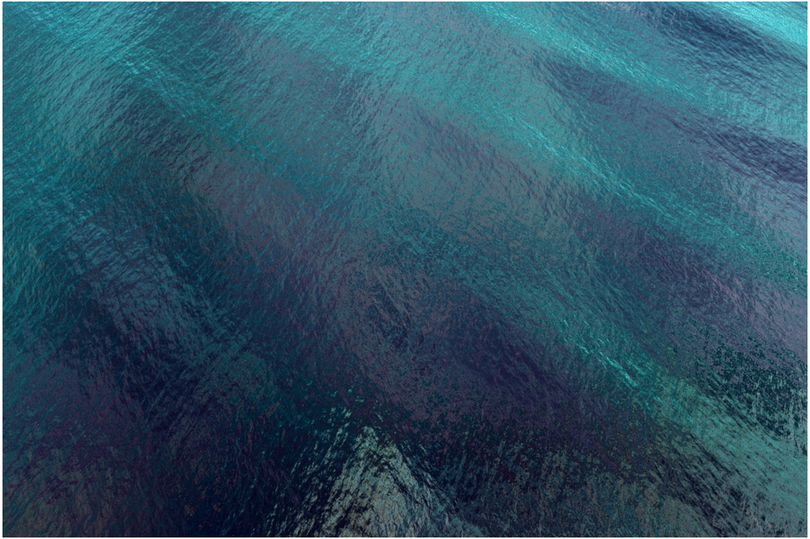
$$\mathtt{eta} = n_1/n_2 = 1.0/1.33$$

- Water $\rightarrow$ air

$$\mathtt{eta} = n_2/n_1 = 1.33/1.0$$

- $\mathtt{color} = \mathtt{color}_{\mathtt{water}} + \mathtt{reflection} * \mathtt{fresnel} + \mathtt{refraction} * (1 - \mathtt{fresnel})$
- Reflection and refraction are looked up in the cube map

FINAL RESULT



ASSIGNMENT 4

- Expand `createQuad()`
  - Increase tessellation
  - Add texture coordinates
- Water shader
  - Use `diffuse.vert` and `diffuse.frag` as guidance
- Resources (`res/`):
  - `textures/waves.png`, normal map
  - `cubemaps/NissiBeach2/`, cube map set
- Files you have to modify (or add)
  - `src/EDAF80/assignment4.cpp`
  - `src/EDAF80/parametric_shapes.cpp`
  - `shaders/EDAF80/water.vert` (add)
  - `shaders/EDAF80/water.frag` (add)

EDAF80:
Seminar 4

DEMO