Assignment 4 — Water Shader

Lund University Graphics Group

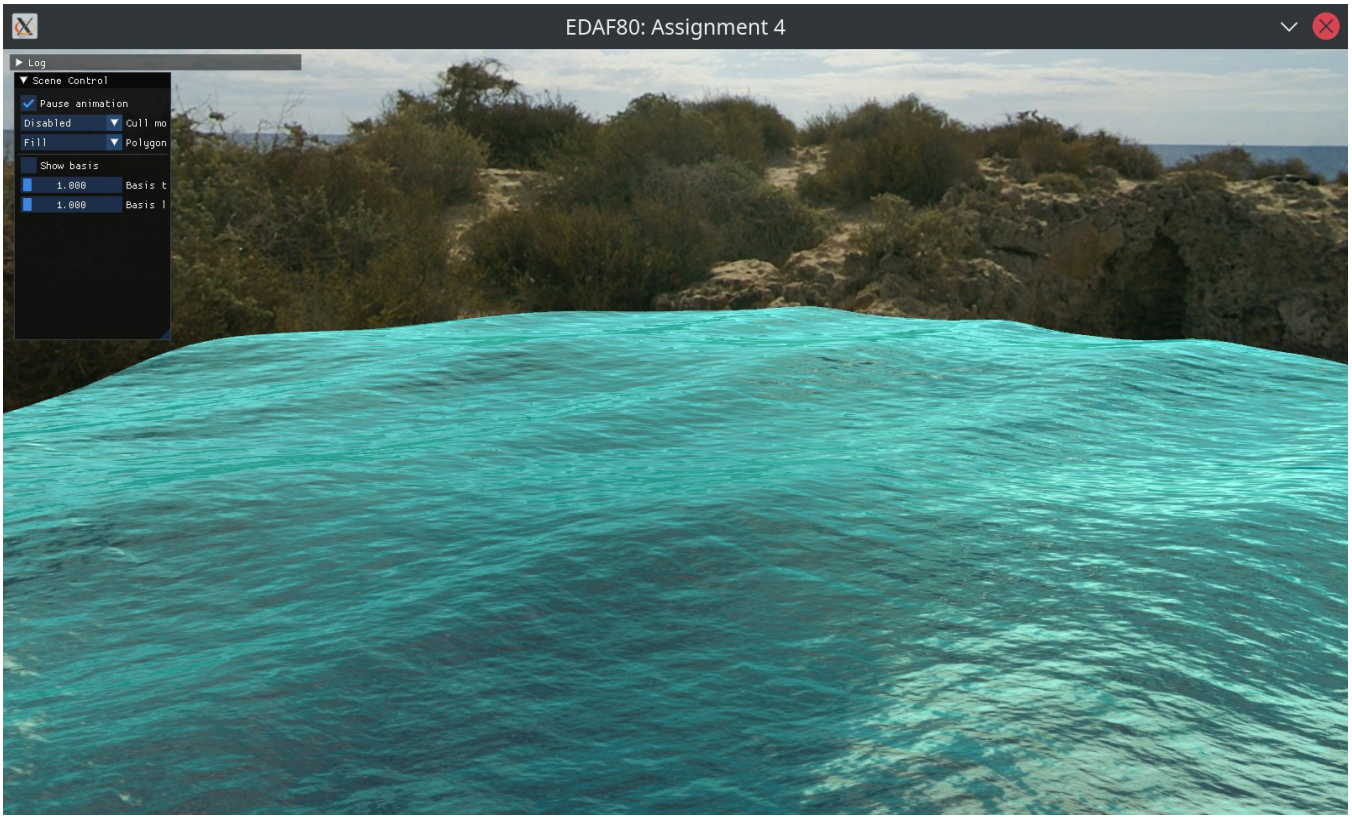

Figure 1: The end result once you have completed everything.

In this assignment you will utilise previously learnt shader techniques to create a water shader. Though fluid mechanics can be simulated accurately by particle systems or FE-analysis, cheap water-like effects can be achieved in real-time using vertex displacements and mapping techniques. Here, you will use trigonometric functions to simulate waves, animated normal mapping for ripples, reflection and refraction etc.

Unlike previous assignments, you will now create the code yourself; you should start from the skeleton code found in `src‣EDAF80‣assignment4.cpp`, but you can refer to `src‣EDAF80‣assignment3.cpp` if needed.

1 Quad tessellation

Before trying to simulate any waves, a higher triangle count than the two currently making up the quad will be needed to create more detailed waves. So the first step will be to tessellate the quad from `parametric_shapes::createQuad()` in `src‣EDAF80‣parametric_shapes.cpp`.

Exercise 1:

1. Create the quad in the x, z-plane rather than the x, y-plane
2. Tessellate it, and generate texture coordinates for it

   Note: normals, tangents and binormals are not needed as those will be computed directly in the shader.
3. Create a quad with a size of 100×100 and a tessellation of 1000×1000 in `src‣EDAF80‣assignment4.cpp`

2 Custom functions in GLSL

You can define custom functions in GLSL similar to C or C++. For example,

```
void myFunction(in float a, out float b,
                inout float c) {
  // b = ???
  b = a + c;
  c = 7.7;
  a = 42.0;
}

void main() {
  float a = 2.2, b = 3.3, c = 4.4;
  myFunction(a, b, c);
  // a = 2.2, b = 6.6, c = 7.7
}
```

As you can see, it is very similar to C or Java but with a few additional constraints:

- there are no pointers or references, instead there are three different parameter qualifiers:

  in a *copy* of the argument is passed to the function; this is the default if no qualifier is used;

  out the variable has undefined content when entering the function, but whatever was written to it by the function will be visible to the caller;

  inout the argument is passed by *reference* to the function, i.e. the function sees whatever was stored in

the variable before being called, and all modifications by the function will be visible to the caller.

- they can not be recursive.

More details can be found in the OpenGL Wiki.

3 Water shader

Implement the features of the water using the theoretical background provided in the seminar and the GPU Gems article.

The equations for a single wave and its derivatives:

$$y = G_i(x, z, t) = A_i \alpha_t^k \tag{1a}$$

$$\frac{\partial G_i}{\partial x} = 0.5 k_i f_i A_i \alpha_t^{k-1} \times \cos\Big((D_{i,x}x + D_{i,z}z) f_i + t p_i\Big) D_{i,x} \tag{1b}$$

$$\frac{\partial G_i}{\partial z} = 0.5 k_i f_i A_i \alpha_t^{k-1} \times \cos\Big((D_{i,x}x + D_{i,z}z) f_i + t p_i\Big) D_{i,z} \tag{1c}$$

with $\alpha_t = \sin\Big((D_{i,x}x + D_{i,z}z) f_i + t p_i\Big) 0.5 + 0.5$, t the time, $\begin{pmatrix} x & z \end{pmatrix}$ the position on the plane of the water surface and the following wave attributes:

A_i the amplitude;
D_i $= (D_{i,x}, D_{i,z})$, the direction of travel;
f_i the frequency;
p_i the phase;
k_i the sharpness.

To combine multiple waves together, the following applies:

$$H(x, z, t) = \sum G_i(x, z, t) \tag{2a}$$

$$\frac{\partial H}{\partial x} = \sum \frac{\partial G_i}{\partial x} \tag{2b}$$

$$\frac{\partial H}{\partial z} = \sum \frac{\partial G_i}{\partial z} \tag{2c}$$

Exercise 2:

1. Create a new shader program; you can use the different *diffuse* shaders as a starting point.
2. Send the elapsed time (stored in `ellapsed_time_s`) in seconds since the program start to the water shader.
3. In the vertex shader, displace the vertices in the y-direction using Equation (2a) for two superimposed waves with the following attributes:

Attribute	Wave 1	Wave 2
Amplitude, A_i	1.0	0.5
Direction, D_i	$\begin{pmatrix} -1 & 0 \end{pmatrix}$	$\begin{pmatrix} -0.7 & 0.7 \end{pmatrix}$
Frequency, f_i	0.2	0.4
Phase, p_i	0.5	1.3
Sharpness, k_i	2.0	2.0

   Note: we recommend defining your wave equation and its derivative as one (as they share quite a few computations) or multiple equations rather than performing the computations directly in the `main()` function and duplicating the code for each wave evaluated.
4. Compute the basic water colour based on the surface orientation relative to the camera (see Figure 2).
5. Add reflection mapping (see Figure 3); you can use the *NissiBeach2* cubemap set.
6. Add wave ripples through the animated normal mapping (see Figure 4); use `res‣textures‣waves.png` as the normal map.
7. Add a Fresnel factor to the reflection lighting.

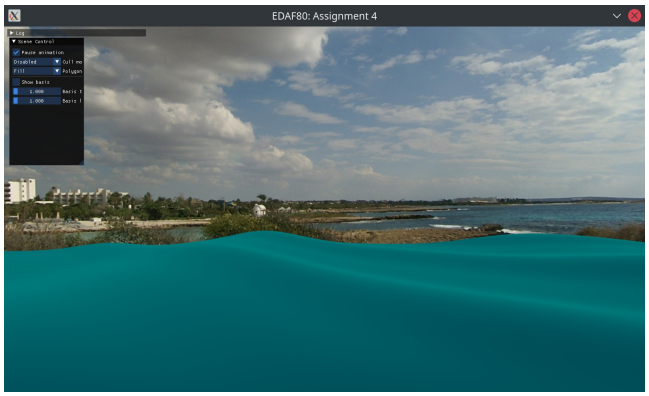

Figure 2: Waves with basic water colour.

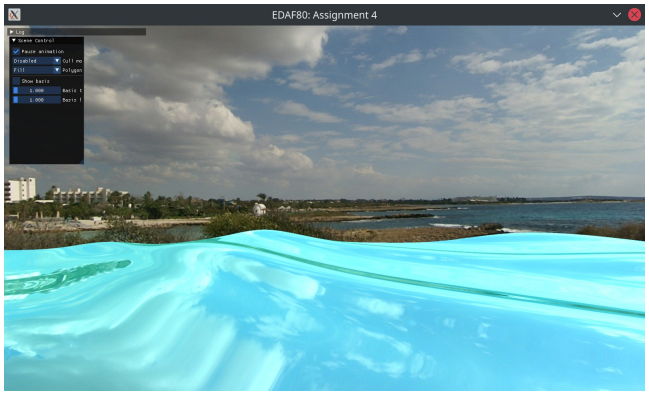

Figure 3: Waves with reflections.

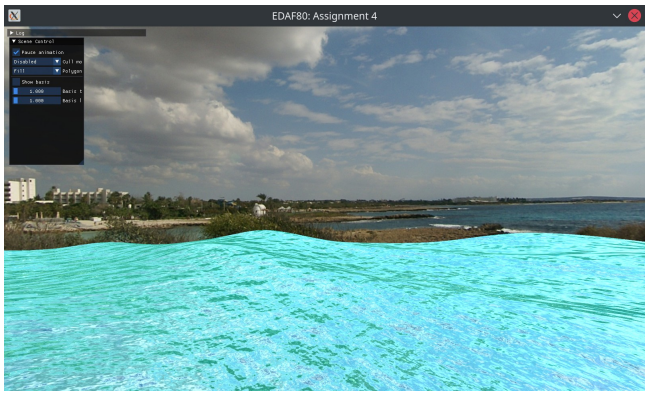

Figure 4: Waves with normal-mapped reflections.

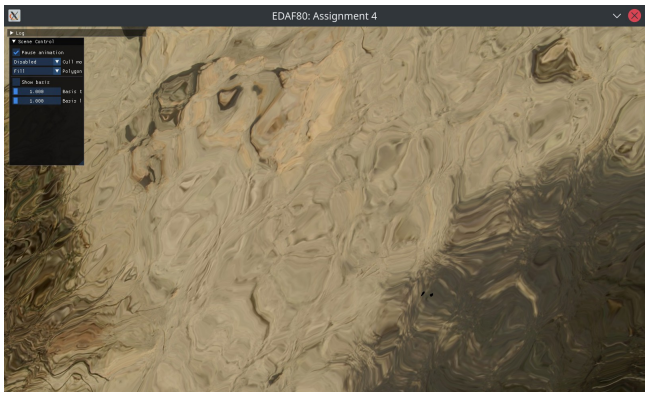

Figure 5: Camera looking straight down the $-y$-axis, with only refractions turned on.

Figure 6: Camera looking straight down the $-y$-axis, once passed the water geometry.

8. Add refraction mapping using an appropriate refraction index. Use the same cubemap set as in Step 5 and take the Fresnel factor into consideration. You should have something resembling Figure 1.

   It can be hard to tell if refraction is working properly, with all the other effects on. So check by setting the final colour to only the results of the refraction mapping, and have the camera look straight down at the water plane. Move the camera close to the water surface, and what you see there (the overall shapes and colours) should match what you see if the camera pushes through the water and looks underneath (still looking straight down the y-axis). For example, Figure 5 showing only the refraction compared to Figure 6 showing the bottom face of the cube map from the same x, z camera position.

4 Suggestions and Thing to ponder

- What do you need to change in order to make the shader work when the camera is below the water? You can use the GLSL variable `gl_FrontFacing` to know whether the current fragment is front- or back-facing.
- Attempt, by tweaking the various coefficients and parameters, to improve the visual appearance of the water even further.
- What would it take to apply this shader to one one of the objects from your previous lab code? If time permits, try it out.

A Framework controls

The framework uses standard key bindings for movement, such as W, A, S, and D. But there are also custom key bindings for moving up and down, as well as controlling the UI. All those key bindings are listed in Table 1.

There is only one action currently bound to the mouse, and that is rotating the camera. To do so, move the mouse while holding the left mouse button.

GUI elements can be toggled being a collapsed and expanded state by double clicking on their title bar. And they can be moved around the window by dragging their title bar wherever desired (within the window).

Table 1: Various controls when running an assignment."Reload the shaders" is not available in assignments 1 and 2 of EDAF80, while "Toggle fullscreen mode" is missing from assignment 2 of EDAN35.

Action	Shortcut
Move forward	W
Move backward	S
Strafe to the left	A
Strafe to the right	D
Move downward	Q
Move upward	E
"Walk" modifier	⇧
"Sprint" modifier	Ctrl
Reload the shaders	R
Hide the whole UI	F2
Hide the log UI	F3
Toggle fullscreen mode	F11

B IDE key bindings

To help with getting certain tasks done more efficiently, Table 2 lists key bindings of different IDEs for several common actions.

Table 2: Various keyboard shortcuts for Visual Studio 2019 and 2017, and Xcode.

Action	Shortcut	
	Visual Studio	Xcode
Build	Ctrl + B	⌘ + B
Run (with the debugger)	F5	⌘ + R
Run (without the debugger)	Ctrl + F5	
Toggle breakpoint at current line	F9	⌘ + \
Stop debugging	⇧ + F5	⌘ + .
Continue (while in break mode)	F5	ctrl + ⌘ + Y
Step Over (while in break mode)	F10	F6
Step Into (while in break mode)	F11	F7
Step Out (while in break mode)	⇧ + F11	F8
Comment selection	Ctrl + K, Ctrl + C	⌘ + /
Uncomment selection	Ctrl + K, Ctrl + U	⌘ + /
Delete entire row	Ctrl + X	