Water simulation in OpenGL

Field of study: Media informatics
Semester: 4

Date: Juni 22, 2018

Authors:
Fabian Niehaus
Tuyet Nguyen

Contents

1 Abstract **2**

2 Wave theory **2**

3 Realisation **3**
3.1 Creating a basic interface with QT 3
3.2 Creating the data structure 3
3.3 Creating the surface mesh 4
3.4 Calculating the mesh height 4
3.5 Rendering 4
3.5.1 Rendering as a wireframe 4
3.5.2 Rendering as an opaque surface 5
3.6 Wave peflection 5
3.7 Dynamic parameters 7
3.8 Full application 8

4 Conclusion **8**

5 Acknowledgements **8**

6 References **8**

List of Figures

1 Circular sine wave (WAVE GIF 2016) . . . 2
2 Complex Wave Equation in Matlab (Hering-Bertram 2018b) . . . 2
3 Circular cosine wave function . . . 2
4 Variable definitions (Wallrath 2005) . . . 3
5 Circular cosine wave function with diminishing amplitude . . . 3
6 Calculating mesh height (pseudo code) . . . 4
7 Rendering as wireframe . . . 5
8 Rendering as opaque surface . . . 5
9 Circular Waves reflecting on a straight barrier (PhysicsLAB 2018) . . . 6
10 Image Source Method (Hering-Bertram 2018a) . . . 6
11 Calculating OPs for reflection waves (pseudo code) . . . 6
12 User Inputs . . . 7
13 Updating the waves (pseudo code) . . . 7

1 Abstract

In this project we will build a C++ application to simulate circular waves on a 3D mesh surface. We will also simulate reflection using the Image Source Method. We will use QT Creator for programming, QT for window management and OpenGL for 3D rendering.

2 Wave theory

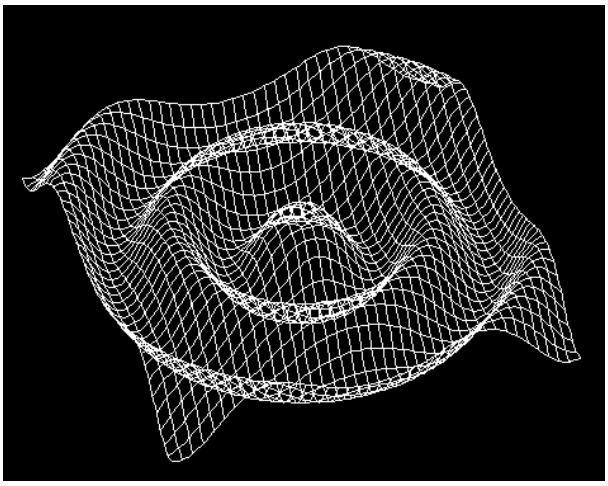

Figure 1: Circular sine wave (WAVE GIF 2016)

In this project, we will use a basic cosine wave with variable parameters for amplitude, wavelength and point of origin (POI). This function is derived from a complex wave function of which we only use the real part.

```
X = linspace( -50, 50, 201); % 201 samples from -50 to 50

% complex wave centered at x=-17, wave length = 8.5
F = exp( i*abs(X*2*pi/8.5)); % first wave
freq = 10;
for t=0:.001:1 % one second in steps of milliseconds
  phi = -2*pi*freq*t; % phase shift due to time delay
  F1 = exp( i* phi) * F; % add phase shift to F
  plot( X, real(F1)); % plot real part
  axis([-50 50 -5 5]); % define figure scale
  drawnow
end
```

Figure 2: Complex Wave Equation in Matlab (Hering-Bertram 2018b)

A cosine wave with circular expansion can be calculated using the formula

$$P(x, z, t) = a * \cos(k * \Delta + \Phi)$$

Figure 3: Circular cosine wave function

where a describes the wave's amplitude, k describes the wave number, Δ describes the distance to the wave's OP and Φ describes the phase shift.

Wavelength:	l
Time:	t
Wave Number:	$k = \frac{2\pi}{l}$
Phase Shift:	$\Phi = -2 * \pi * f * t$
Frequency:	$f = \frac{c}{l}$
Phase Velocity:	$c = g * l * 2\pi$
Gravitational Acceleration:	$g = 9.81 m/s^2$

Figure 4: Variable definitions (Wallrath 2005)

As this project aims for a somewhat realistic behaviour of the wave, the amplitude needs to diminish with increasing distance to the wave's point of origin. In order to accomplish this effect, the distance to OP is factored in a second time:

$$P(x, z, t) = a * \cos(k * \Delta + \Phi) * \frac{1}{\Delta + 1}$$

Figure 5: Circular cosine wave function with diminishing amplitude

As Δ equals 0 at the wave's OP and multiplier needs to always be less than 1, 1 is added to Δ in the multiplier's divisor.

Using this formula, the height for every point of a mesh can be calculated. This formula can also be used to calculate multiple overlapping waves. In that case, the new height for a point is the sum of all waves.

3 Realisation

3.1 Creating a basic interface with QT

First off, we create a new QT widget application. This allows us to use QT Creators design feature to set up our application's interface. A new QOpenGLWidget is placed and will be used as a placeholder for a new custom class inheriting QOpenGLWidgets functionality. This class, called OGLWidget, needs to implement the following methods: initializeGL (for setting up OpenGL), paintGL (for doing the actual rendering), resizeGL (for handling resizes of the display window). Additionally, the functions stepAnimation SetMaterialColor and InitLightingAndProjection[1] are used.

3.2 Creating the data structure

The data structure is separated in different classes.
The class "Wave" contains a single wave's parameters such as wavelength l, amplitude a and point of origin Δ. It also holds the values for wave number k, phase speed c and frequency f, which are calculated when the object is instanciated.

[1]Taken from Prof. Dr. Martin Hering-Bertrams OpenGL_Example

The class "Wavesurface" contains the surface mesh as well as the logic for calculating the mesh height.

3.3 Creating the surface mesh

The surface mesh consists of a two dimensional list of QVector3Ds, QT's own implementation of a three value vector. These vectors store a single point's x, y and z coordinates. We settled on a dimension of 50 x 50, which is sufficient from both a graphical and a performance point of view.
The mesh is located in the x-z-plane, meaning the waves' 'height' is calculated in y-direction.

3.4 Calculating the mesh height

Using the formula $P(x, z, t) = a * \cos(k * \Delta + \Phi) * \frac{1}{\Delta+1}$ explained under Wave Theory, a function which calculates the sum of amplitudes for all waves for every point of the mesh is implemented.

```
for each mesh row
  for each mesh column
    double height //Hoehe
    for each wave
      add result of formula stated above to height
    set y of current Point to height
```

Figure 6: Calculating mesh height (pseudo code)

3.5 Rendering

In order to display the mesh on the screen, OpenGL needs to be instructed to render the points. Before rendering, the original coordinate system is rotated by 45Âř in positive x and negative y direction so the mesh is visible from an angle. Then, the specific rendering methods are called.

3.5.1 Rendering as a wireframe

First, the mesh is draw as a wireframe using the method drawMeshWireFrame(). This function connects all mesh points into triangles and the draws them on the screen.

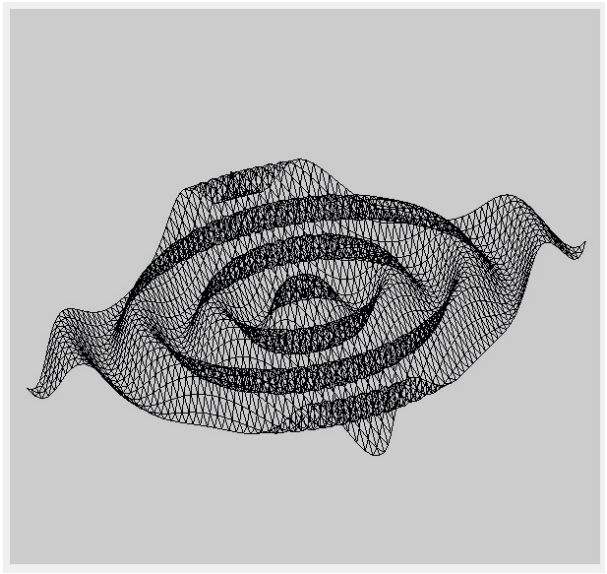

Figure 7: Rendering as wireframe

3.5.2 Rendering as an opaque surface

After drawing the object as a wireframe we want to draw it as an opaque surface with lighting. This is being achieved in the method drawMeshQuads() which connects all points into quads. This time using GL_Quads, the four vertices of a quad are connected and the area inbetween is filled. The normal vector for this is calculated using the cross product of the two diagonals vectors.

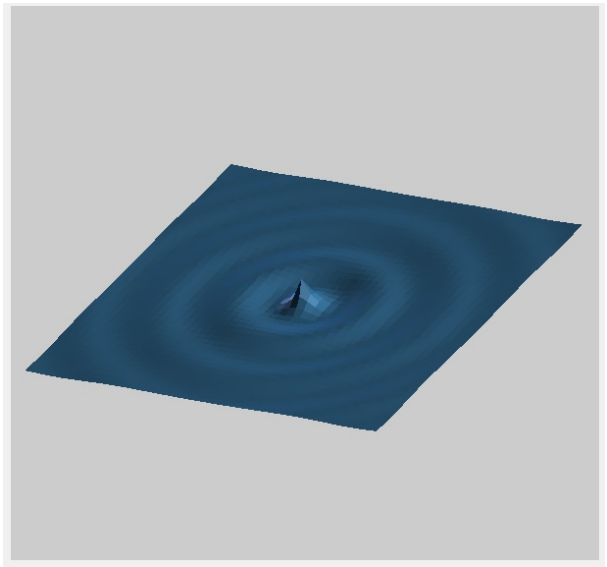

Figure 8: Rendering as opaque surface

3.6 Wave peflection

In this project, we also aimed to simulated reflection of the waves as if our mesh was a body of water located inside a rectangular container.

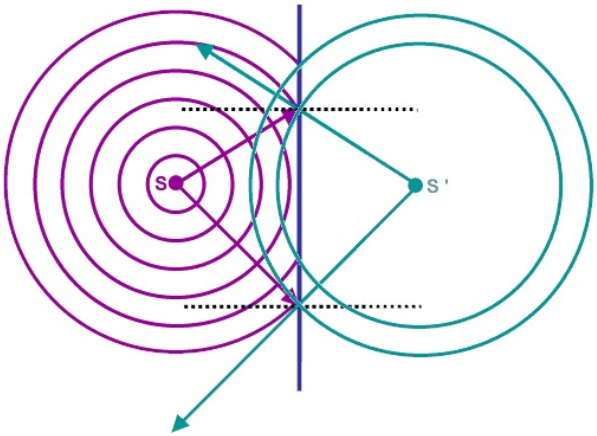

Figure 9: Circular Waves reflecting on a straight barrier (PhysicsLAB 2018)

In order to achieve this effect, the so called Image Source Method was used. This method simulates reflection by placing another wave on the other side of the felection obstacle. The OP of this additional wave is placed at the same distance to the obstacle that the original wave's OP is at.

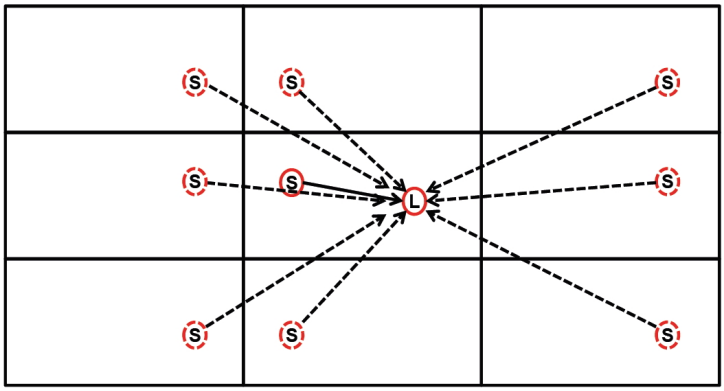

Figure 10: Image Source Method (Hering-Bertram 2018a)

To ensure an acceptable level of reflection, we placed an additional wave in each of the eight neighboring tiles. This way, the original wave is reflected from each direction.

```
meshX = dimension of mesh in x direction
meshZ = dimenstion of mesh in z direction
origX = x value of original wave's OP
origX = z value of original wave's OP

w1_Origin = (-1 * meshX - origX, mesh/ - origZ); // links oben
w2_Origin = (origX, mesh/ - origZ); // oben
w3_Origin = (meshX - origX, mesh/ - origZ); // rechts oben
w4_Origin = (-1 * meshX - origX, origZ); // links
w5_Origin = (meshX - origX, origZ); // rechts
w6_Origin = (-1 * meshX - origX, -1 * mesh/ - origZ); // links unten
w7_Origin = (origX, -1 * mesh/ - origZ); // unten
w8_Origin = (meshX - origX, -1 * mesh/ - origZ); // rechts unten
```

Figure 11: Calculating OPs for reflection waves (pseudo code)

3.7 Dynamic parameters

In the last step, user inputs for dynamic parameters were implemented. The following wave and program parameters can be set at runtime:

- Amplitude
- Wavelength
- Point of origin
- Reflection (On / Off)

Inputs are made via sliders for numerical values and a checkbox for on/off values.

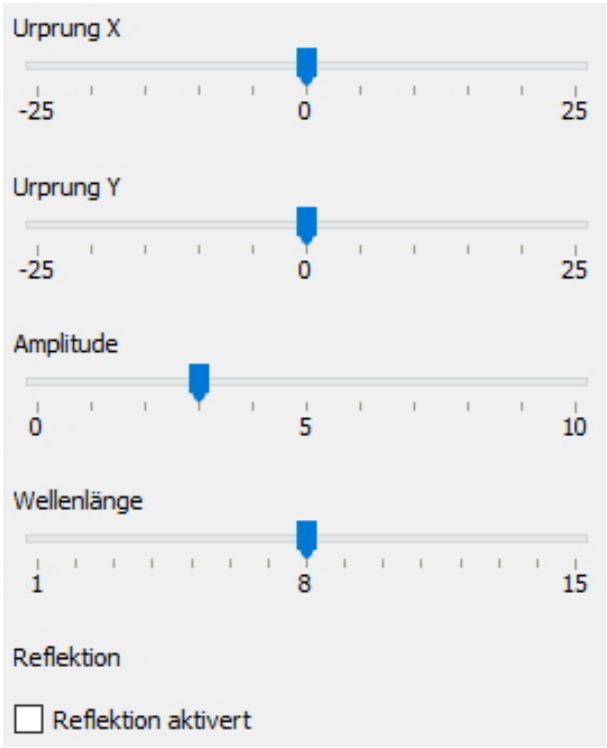

Figure 12: User Inputs

Every slider is connected to a corresponding function with the program using QT's SIGNAL-SLOT mechanism (compare to Sliders Example, The QT Company 2018). For example, the slider for the OPs x coordinate is connected to the updateWaveX() method of the program.
Whenever an input is changed, the updateWaves() function is called from the connected funtion. Every value except the changed one is copied from the current state and all waves are subsequently deleted and then newly created with the new parameter.

```
delete all waves
create new origin wave (amplitude, wavelength, OP)
if relfection is active
  create the 8 corresponding reflection waves
```

Figure 13: Updating the waves (pseudo code)

3.8 Full application

The following application is the functional result of implementing all aspects listed above:

4 Conclusion

We were able to create a simple wave simulator with wave reflection and user inputs. We gained deeper understandig of the mathematics involved in wave calculation and reflection.
Further improvements to the simulation could be made by using so-called Gerstner waves instead of cosine wave. These waves resemble the actual behaviour of water more closely. The mesh calculations could also be offloaded into the GPU via the use of shaders. Shadres would also allow for more realistic display of water, including refraction and skybox reflection.

5 Acknowledgements

- The Wolfram Alpha computational engine for helping out with some of the mathematics involved
- The QT Company, GitHub, the developers of TeXStudio and the developers of MiKTEX for letting us use their software free of charge
- Prof. Dr. Martin Hering-Bertram for providing the OpenGL example code for coloring, lighting and projection used in the project

6 References

Sources

Hering-Bertram, Martin (2018b). *Simulated Acoustics (from Computer Graphics 2018)*. Hochschule Bremen.

Sliders Example (2018). The QT Company. URL: http://doc.qt.io/qt-5/qtwidgets-widgets-sliders-example.html (visited on 06/25/2018).

Wallrath, Timo (2005). "Simulation und Visualisierung von Wasseroberflaechen". Universitaet Koblenz und Landau.

Pictures

Anonymous (2016). *WAVE GIF*. URL: https://giphy.com/gifs/wave-L4PNHiFUAYa9a (visited on 06/28/2018).

Hering-Bertram, Martin (2018a). *Image Source Algorithm (from Computer Graphics 2018)*. Hochschule Bremen.

PhysicsLAB (2018). *Circular Waves Reflecting off of a Straight Barrier*. URL: http://dev.physicslab.org/img/259fff4f-b7e5-4b2d-81f9-c3eda829e407.gif (visited on 06/28/2018).