Oil-Spill Simulation Using Bi-Layer Shallow Water equations

Utrecht University

Department of Information and Computing Sciences

Master Thesis - Game and Media Technology Program

Utrecht University

Supervisor:
Dr. A. Vaxman

Student:
T. Percheul

May 2017

Abstract

We describe a novel algorithm to simulate oils spills on open oceans, as a two-layer fluid simulation with the Shallow Water Equation (SWE). Our algorithm simulates spreading and advection, as the main driving forces of oil spread, while also attending to emulsification and evaporation, as secondary driving forces. We provide a solver using the two step Lax-Wendroff method, that allows the algorithm to run efficiently and in parallel on the GPU.

Contents

1 Introduction

Oil spills are primarily man-made catastrophes that have a severe negative impact on the long-term natural and economical resources of large regions for decades. In order to limit the impact of such accidents, human intervention can help collect and clean the *slicks* (spill of oil). However, proper training for such procedures is expensive (need of maritime resources, collection instruments, etc.) and cannot faithfully simulate the correct effect in the required scale. Computer-based simulation naturally presents itself as a cost-effective alternative, permitting representation of different and realistic scenarios, with high reproducibility and flexible customization to the problem at hand.

We present a novel way to simulate slicks, making it possible to design trainning software involving boats, changing environemental conditions (wind, current, shores and other water limits, etc.) and cleaning forces (booms, skimmer, burning, chemical cleaning, etc.).

1.1 Background

The work described in this thesis is done in collaboration and supervision of VSTEP. The core activity of VSTEP is the production of simulations for professional trainning (Figure 1). They produce the "Nautis 3 Maritime simulator" for recreating plausible maritime scenarios in the perspective of training and educating pilots, mates and other vessel operators. The Nautis3 has the following features: realistic sea state and weather, radar reading, navigations of ships with accurate physical controllers. In particular motors for navigation and steering act realistically. The Nautis 3 offers also a wide range of ships and environments, from container ships in the open ocean, through tug boats in the harbour of Hong-Kong, to péniches in the canals around Rotterdam.

The given assignment by VSTEP is to create an oil-spill simulation to be integrated inside the Nautis simulator (Realistic Maritime Simulator used by civilian and military professionals). The oil spills are required to behave realistically (considering all the physical and weather-based effects), respond to user interaction (e.g., a boat could part the slick, as in a real spill) and most importantly, run in real time; the algorithm is meant for real-time training, and thus has to be highly efficient. This mandates a highly-parallelizable formulation that can run efficiently on the GPU.

A training session takes about around two hours. A supervisor builds a scenario for the trainees beforehand. The scenario comprises the amount of oil and its sources, the elapsed time since the spill, the available boats and resources and the other environmental hazards. The trainees manoeuvre the boats and their resources to curb the progression of the slick and clean it. Therefore, the ease of setting up a spill should also be taken into account (adding oil with a click, or changing its "age" and properties through a graphical user interface).

Figure 1: One of the possible NAUTIS trainning station from VSTEP (here for oil tankers) and a screnshot of the NAUTIS software during a tugboat trainning.

1.2 Motivation and Challenges

There are many technical challenges in setting up a simulation, and in this thesis we focus on some of the challenges in the faithful discretization of physical properties of slicks. They are as follows:

- Simulation of two non-miscible (non mixing) fluids, such as oil and sea water, is not well-studied, especially for simplified models to be used in real-time;
- The density and viscosity of oil spills changes over time, and there is (very) little work considering this aspect;
- State-of-the-art methods for oil spill are more oriented towards non-interactive long-term simulation (i.e., weather forecasting), whereas there is an increasing demand for real-time simulation of oil spills and cleaning.

1.3 Research Question

We provide solutions to the following research questions:

- How to discretize and represent the important elements of an real time oil spill simulation?
- Can we provide a discretization, and consequent simulation, that would be parallelizable and cost effective?
- How should we model the simulated fluid to reflect oils with different densities and properties?

1.4 Contribution

Our main contributions are:

- A highly parallelizable, GPU intensive algorithm, based on the Shallow Water Equations(SWE), for simulating bi-layer, non-miscible, viscous fluids.
- Introducing a novel grid refinement technique based on automatic mesh refinement (AMR) that improves efficiency.

2 Background Research: Fluid Simulation

Fluid Simulation is an extensively researched topic. It is popular for various applications, ranging from entertainment (fluids for games and films) to physical modelling and measurement (e.g. astrophysical phenomenon and cellular level modelling). There are several approaches to fluid simulation, and each has its own advantages and disadvantages. Typically, the ultimate goal of these simulations is to provide an approximation and discretization to the Navier-Stokes equations (see 3.1.1). Such approximations may be achieved by modelling the entire fluid (3D grid-based or particle-based methods), or reducing the problem to model only the surface of the fluid (e.g., with height fields).

There are two major approaches for the discretization and representation of fluids in a simulation: Lagrangian and Eulerian. Lagrangian methods track packets of material as they move through space, and Eulerian methods track changes at fixed points over time (e.g., on grids. See Figure 2). Broadly speaking, the difference between these approaces can be explained as follows [BMF07]: for a bridge over a river, the Lagrangian method is equivalent to sitting in a boat and making measurement of the drifting, whereas the Eulerian method is equivalent to remaining on the bridge and making measurement of the water passing under it.

We recommend [GDC08a] and [GDC08b] for a review of the (serious) game-industry preferred algorithms for fluid simulation and [MSJT08] for a deeper review.

In this section, we review state of the art fluid simulation algorithms, and see how the computational cost or lack of realism is limiting us to simpler methods such as 2D Euleurian grids

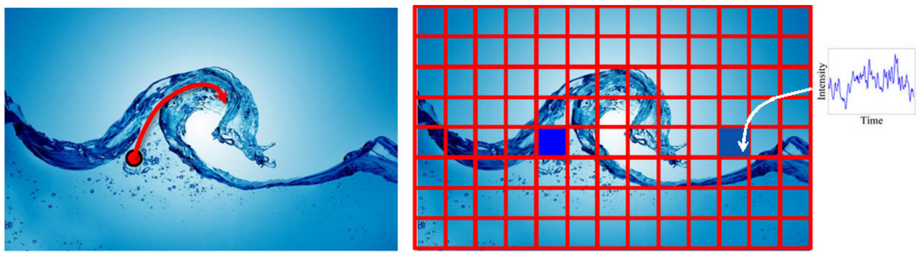


Figure 2: Lagrangian vs. Eulerian representation of fluids: one tracks spatial changes of a point, the other one changes at a fixed point in space.

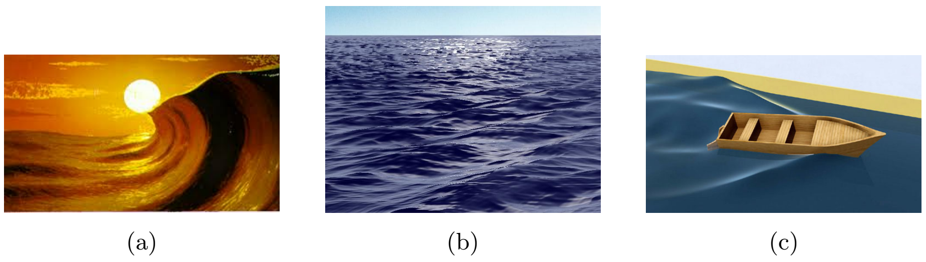
2.1 Procedural Water

Procedural water is the implicit simulation of the surface of a fluid body (usually oceans, or lakes, but can apply to any fluids) in 2D (domain reduction to reduce simulation costs) using groups of sine waves whose interference creates the waves. The waves are formed by combining a set of sine waves, that interfere in an certain region of space, making a wave train (or wave packet) [FR86](Figure 3a). The simulation of several train waves with different amplitude, speed and wave numbers over the body, creates a realistic agitation of the sea. This simulates different types of waves such as open sea gravitational waves ([HNC02], Figure 3b) breaking waves ([FR86]) and even waves generated from interaction with objects ([YHK07] & [BHN07], Figure 3c).

This type of simulation offers a broad creativity for animations: a animator can add as many waves as wanted, as long it fits his view of the fluid. However, for realistic simulations, specific sets of waves have to be given such as the ones in [FR86] .

Having breaking waves is very simple using this model, as detecting the breaking waves is straightforward [FR86, HNC02]. Most of the remaining work lies in visually representing the break.

An important component of oil-spill simulation is managing the spreading. Above papers only consider unbounded water bodies or fixed boundaries, and focus on the simulation of waves on such bodies. The simulation also requires the oil slick to have an expanding body (whilst tracking volume). As such, not only interaction effects need to be present, but also the spreading behaviour. This requirement is sufficient enough to consider another modelling algorithm.



(a) (b) (c)

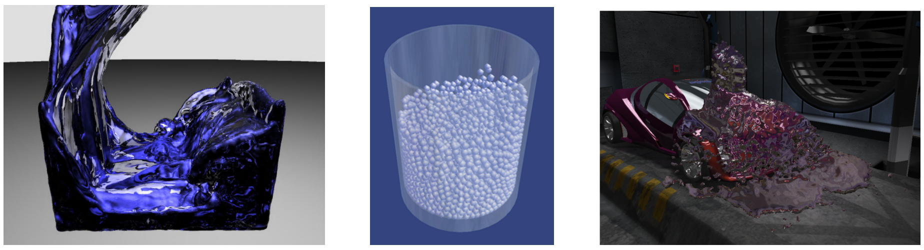
Figure 3: Procedural waves from [FR86], [HNC02] and [YHK07] respectively, Very suitable for animation, harder to use to expanding 3D fluids

2.2 Particle Based Fluids

Particle-based simulations are straightforward, but naïve. In nature, liquids are composed of billions of particles interacting with each other simultaneously. As a computer cannot hope to process so much information, some trade-offs are made : every simulation particle represents a fraction of the fluids (a packet of molecules) that interacts with its neighbours in every frame.

Many particle simulations are based on the Smoothed Particle Hydrodynamics method (SPH) from [Mon92], more recently implemented by [PTB$^+$03] (Figure 4a). In this method, the positions of the particles are tracked (and thus this is a Lagrangian model). As this was originally developed for simulating large astrophysical events, the method is based on distance-based kernels that model the effect of particles on each other, permitting the addition of different effect one would like to apply like gravity, external forces, magnetism or even space-time deformation.

Interactive, real time simulation can be obtained ([KW06] & [MCG03], figure 4b), but it is limited in the number of simulated particles, restricting to small to medium simulations ([MCG03] works with barely 5000 particles), and therefore ruling it out for a slick on open ocean simulation.



(a) (b) (c)

Figure 4: Particle fluids from [PTB$^+$03], [MCG03] and [MSD07] respectively, we can observe the incredible level of detail, but it forbids real time computations.

2.3 Eulerian 3D Grid

Fluids can be modeled using 3D grids, in the Eulerian method. Conversely to Lagrangian methods, they track the behavior of the fluid in a fixed point on a grid. These Eulerian grids have the capability to track many different effects like splashing, but are limited in their simulation quality by their grid resolution. Recent work like [CM11] (figure 5a) use varying height cells to efficiently simulate complex effects by focusing the quality of the simulation at the surface (a finer grid is used at the surface for more fine-grained effects, and a coarser grid at the bottom for the correct volume advection).

There are also several hybrid methods that combine Lagrangian and Eulerian elements for efficiency, such as [TWGT10], by using a grid for the general advection, and a particle-based method for fine elements (such as droplets). There are also implementations to improve the quality using a surface tracker [BHW13] (figure 5b), spatio-temporal extrapolation [ZM13] (figure 5c) or a coarser grid with progressive refinement [LZF10].

3D grids can prove very effective for fluid simulations, but oil spills occur mostly on the surface of the fluid, and thus simulating the entire grid can be wasteful.

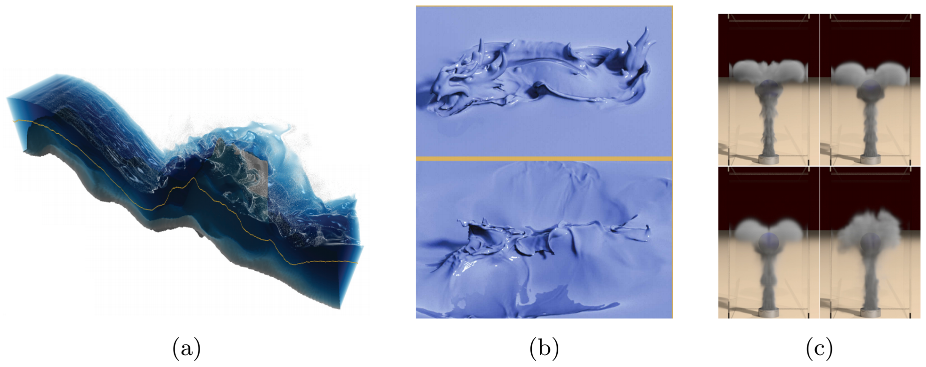


Figure 5: 3D Eulerian methods from [CM11], [BHW13] and [ZM13] respectively, 3D grids permits high level of details, and usage of different quality grids can vary the realism as for [ZM13].

2.4 Eulerian 2D Grid

Following the same principle of the 3D Eulerian grid, 2D grid methods track the fluctuation of a whole column of fluid. The correct equations for representing the fluid transfer from a column to another are derived from an adaptation of the Navier-Strokes equations (discussed in greater details in Section 4).

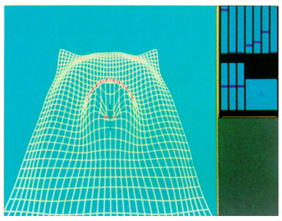
This method is very appreciated in simulations handling fluids alongside other elements, as in videos games (containing physical objects,). They require this cost-effective method to allow computation of other events (physic system, characters, etc...). The *shallow water equations* (SWE) are one of the latest algorithms used by the industry, as it can take into consideration varying topography while staying fast, lightweight and allowing easy interaction[MSJT08] (Figure 6b).

[MSJT08] provides an implementation of the basic SWE solver, and [OH95], [KM90], [TMFSG07] and [CM10] provide together several minor improvement for realism such a small scale details, foam formation, splashing, breaking waves and immersion of solid objects (Figure 6a and 6c). Our Solution takes mainly from the work of [MSJT08] for its simple implementation and ease of modification, but differs by allowing multiple fluids at the same time.

2.5 Conclusion on Fluid Simulation

Considering all the algorithms available for fluid simulation, we have to reject procedural animation techniques, as they do not allow us to model easily expanding fluids, even less when considering a changing topography over time (the "topography" being the surface of the water where the oil rest).

Particle-based methods offer incredible realism, but are too expensive for real-time simulations, where the fluid simulation should leave computation resources available to other game elements (for the NAUTIS to simulate boats and other physics events). However, they can still be used for hybrid techniques as in [TWGT10] for having both advantages of Lagrangian and Eulerian methods (that is breaking waves and splashes).


(a)

(b)

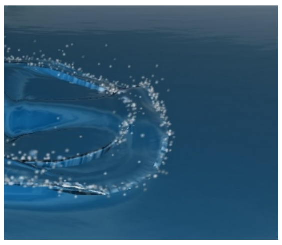

(c)

Figure 6: 2D Eulerian methods from [KM90], [MSJT08] and [TMFSG07] respectively. Early implementations of the grid permitted tracking acurrate changes ([KM90], or [MSJT08] for the more modern version) and later improvement as [TMFSG07] incorporate small scale details re-enforcing realism.

We thus use Eulerian methods, but as we model a flat expanding fluid, it makes sense to reduce the costs and go for 2D Eulerian grid simulation. More precisely, the Shallow Water Equations, as they are well researched and favoured in real time simulations in the game industry (and other industry asking real time realistic simulations) [MSJT08]. No particular research directly treats two layers of non-miscible fluids (at the exception of [DL78] & [ZLO13] who look at a mathematical solution to the problem in 1D), it is thus a contribution of our work to provide a much-needed solution to this problem.

3 Simulating Oil Slicks

We present the principal physical laws governing the behaviour of oil slicks, in order to provide a background to simulating them. We define the discretization we use to simulate these effects in 3.3.

3.1 Oil Slick Weathering

Weathering concerns all the changes occurring from oil water interaction. They are depicted by Figure 7.

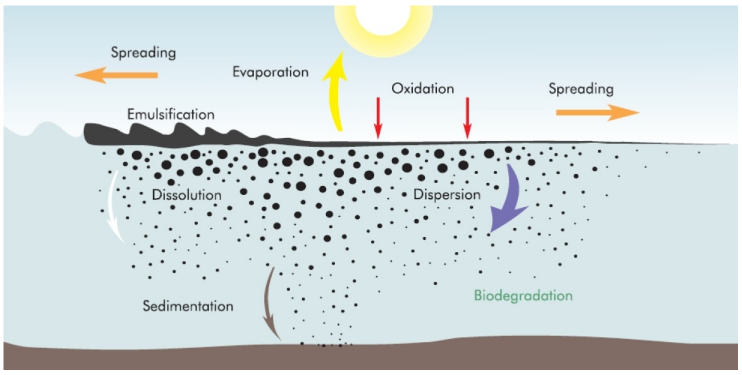


Figure 7: A schematic depiction of all physical phenomena taking place in an oil spill. Some of them are negligible enough to ignore during a real-time real time simulation.

3.1.1 Spreading

Upon contact with water, the oil starts to spread. The model proposed by [Fay71] defines a formulation for oil-spread estimation (see Appendix A). It states the three main spreading ways of a slick: the inertial spread, the viscous spread, and the surface-tension spread. We focus solely on the viscous spread. We do so since, according to [LFBC84, SS95], inertial spreading is only meaningful in the first few seconds, and the surface tension spread is meaningful only after a few months.

We can thus model and simulate the expansion of the oil as a viscous fluid simulation, under viscous spread. These spreading forces are modelled by the incompressible Navier-Strokes equations [BMF07]. They are as follows:

$$\frac{\delta \vec{u}}{\delta t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + v \nabla \cdot \nabla \vec{u} \tag{1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2}$$

Where ρ as the fluid density (kg/m^3), p is the pressure (Pa), $\vec{g}$ represents the external forces such as gravity and v is the kinematic viscosity $(m^2/s\ )$.

(1), called the "momentum equation", is actually the combination of three equations:

- The "Material derivative", represent the derivative of the property of a material point as it moves with time. it is written:

$$\frac{Dq}{Dt} = \frac{\delta q}{\delta t} + \vec{u} \cdot \nabla q = 0 \tag{3}$$

  With q the element we are tracking; it could be a packet of matter or temperature of the fluid, but in our context, it is the velocity of the fluid.

- Gravity and other external forces as $\vec{g}$. Gravity is represented with a downward force applied evenly at any point of the fluid, resulting in a progressive smoothing of the slicks.

- Internal forces, divided in 2 :
  - $\frac{1}{\rho}\nabla p$ represents the effect of hydrostatic pressure (the pressure that the fluid has on self);
  - $\nabla \cdot \nabla \vec{u}$ is the viscosity resistance. Viscosity resistance can be seen as a combination of surface tension, inertia and friction. It represent the resisting motion from oil interaction with water (friction), the resistance to changes of state (inertia) and the cohesive force of the fluid to itself (surface tension). Viscosity is usually omitted in most of the simulations, whereas small errors from computations are generally perceived as the fluid viscosity. However, we do use viscosity for oil slick simulation..

The divergence free condition (2) ensures the conservation of mass, and the stability of the simulation as a whole.

3.1.2 Advection

Advection is the drift of the oil spills due to the surface current and wind. It is computed in [NKAS08], and corresponds to:

$$\vec{V_d} = \alpha_w \vec{V_w} + \alpha_c \vec{V_c} \tag{4}$$

α_w is the *drift factor* (usually between 0.02 and 0.04 [SS95]). $\vec{V_w}$ is the wind vector above water, α_c is the advection factor of the water current (typically 1.1) and $\vec{V_c}$ is the surface current vector. Note that according to [SS95], surface current might be disregarded in open sea where the surface currents tend to be negligible.

3.1.3 Evaporation

[Fin99] provides a detailed evaporation model for most common oils. Typically, oil loses 25-30% of its mass during the first 24 hours. The vapours are mostly the lighter component of the oil, leaving a denser and more viscous slick. Note that unlike water, oil is not strictly boundary-layer regulated, meaning that wind and other factor (except temperature) have very limited influence on the evaporation we can then practically consider the rate of evaporation as constant, if the temperature is considered as constant as well.

We can thus describe the evaporation process with two simple equations, depending on the type of oil (see [Fin99][Table 2] for the correct equation per Oil type).

For oils following a *logarithmic* equation:

$$\text{Percentage Evaporated } E = [.165\,(D) + .045\,(T - 15)]\,ln\,(t) \tag{5}$$

For oils following a *square root* equation:

$$\text{Percentage Evaporated } E = [.0254\,(D) + .01\,(T - 15)]\,\sqrt{t} \tag{6}$$

Where D is the percentage (by weight) distilled at 180°C, T the temperature (°C), and t the time in minutes.

[SS95] also provides complementary equations to calculate the impact of evaporation on the density ρ and viscosity μ parameters of the oil:

$$\rho_t = \rho_{oil} + C_3(E_t)$$
$$\mu_t = \mu_{oil} \times exp\,[C_4 E_t]$$

C_3 and C_4 are constant paremeters depending on the type of oil. C_3 is the mousse viscosity constant (the maximal water content, assumed around .7 for crude/heavy fuel oils and .25 for lighter oils) and C_4 being between 1 and 10 with 1 for gasoline and 10 for crude oil. Both constant values can be found in [MBMP80]

3.1.4 Dispersion

Dispersion refers to the process of oil droplets forming and transferring into water columns due to the turbulences in the sea. This process is influenced by the oil layer thickness, the breaking-wave energy (waves breaking in the slick) and the water temperature [DS88]. Natural dispersion can be separated in 3 phases: globulation, where waves crashing in the slick form oil droplets; dispersion, where the energy given by the waves coupled with the rising forces spreads the droplets; and coalescence, where droplets rise back and merge with the slick [SS95]. However, this quantity is very small ($> 1mg/L$), it can thus be neglected in the simulation.

3.1.5 Emulsification

Emulsification is the process of water droplets mixing into the oil, creating a sort of "oil mousse" emulsion. The incorporation of water increases the viscosity and volume of the slick [NKAS08] and makes it harder to clean (burning becomes impossible, and dispersion or recovery more complex) . The main factor that determines the stability and potential amount of an oil emulsion is the amount of surface active agent (surfactant). It is generally acknowledged that the level of asphaltene determines the ability to emulsify, and the concentration of wax increases the stability of the oil emulsion [SS95]. The water incorporation can be modelled over time as follow [MBMP80] & [SS95]:

$$Y_t = C_3 \left[1 - \exp\left(\frac{-2 \times 10^{-2}}{C_3}(1+W)^2 t\right)\right] \tag{7}$$

Where Y_t is the relative quantity of water, W is the wind speed ($m \cdot s^{-1}$) and C_3 is the mousse viscosity constant.

[SS95] also provides the changes of density ρ and viscosity μ using the Mooney equations:

$$\begin{aligned}\rho_t &= \rho_{oil}(1 - Y_t) + \rho_{water}(Y_t) \\ \mu_t &= \mu_{oil} \exp\left[\frac{2.5Y_t}{(1 - C_3 Y_t)}\right]\end{aligned} \tag{8}$$

With ρ_{oil} and μ_{oil} the base oil density ($kg.m^{-3}$) and viscosity (cP). μ_{oil} can be obtained using [BH88] methods, $\mu_{oil} = 224A^{1/2}$, with A being the asphaltene content (%) of the oil.

3.1.6 Dissolution

Although few papers cover the large-scale phenomenom, [SS95] estimates that roughly 1% of the oil dissolutes very shortly after contact with water. The lighter components of oil are also the first ones lost through evaporation, a process which is 10 to 1000 times faster [Spa88]. Essentially, this process is more relevant to toxicological studies (impact on marine life) rather than slick spread simulation, and can thus be neglected for a cleaning training simulation [SS95].

3.1.7 Photo-Oxidation

Photo-oxidation is the slow reaction of oil with ambient oxygen, either breaking down in soluble products, or forming persistent hard compounds called tars. As suggested, this process is exacerbated by sunlight, but very slowly, and can take weeks/months to be noticeable [Spa88]. We thus also neglect it from our model.

3.1.8 Sedimentation

Sedimentation is the sinking of oil component due to an increase in density that compromises their buyonancy. This happens due to the compsumtion by the marine life or due to fixation of oil on the surroounding suspended matter [Spa88].

3.1.9 Biodegradation

Biodegradation is the consumption of the oil components by the local marine life that rely on hydrocarbons as a source of energy (mostly micro-organism that one would also find near natural cold seep / hydrothermal vents) [Spa88]. Biodegradation occurs along the weathering process and is exacerbated by other weathering mechanisms, as the micro-organism can only consume oil from the oil-water interface. This is nevertheless a very slow mechanism and effects take years to be seen, we thus do not model it.

3.2 Oil Slick Cleaning Process

Cleaning of an oil spill is done in different procedures, the common procedure is to first contain the slick and then collect or dispose of it. Here are the principal methods used for cleaning:

Containment Booms are floating barriers that prevent spreading and scattering of the oil (figure 8b). This permit to contain and allow a certain thickness of oil layer for later treatment.

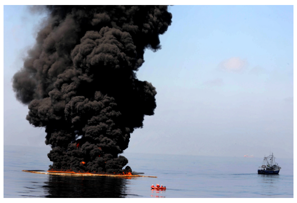
Chemicals are used mainly when there is turbulence preventing the usage of booms. Different chemicals have different effects on the oil. Solidifiers are used to solidify the oil to be collected, and dispersants break down oils to include water and allows it to sink (and be degraded by micro-organisms).

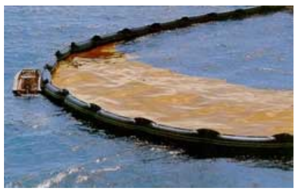
Sorbents are insoluble materials that collects the oil by absorption. They can be later reprocessed (extraction of the oil and reuse of the sorbent).

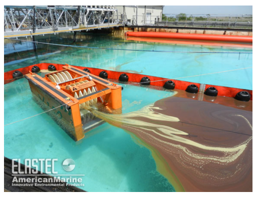
In Situ burning is used when the water is calm and the oil slick contained by booms (figure 8a). Burning the slick is not environmentally friendly, but saves costs.

Bioremediation is the introduction of oil-consuming bacteria or degrading enzymes to accelerate the bio-degradation of oil.

Mechanical removal is done using pumps or skimmers (rotating discs) to separate the oil from the water (figure 8c).


(a)


(b)



(c)

Figure 8: Some cleaning examples: (left) In Situ burning, (middle) slick containment by booms and (right) mechanical skimming of the oil.

3.3 Physical Phenomena in Our Environment

Considering the requirements for simulation and the time span of a training session (Section 1.1), we can factor the behavior of our model into 3 parts:

Modelled in real time: these include advection, spreading, interaction with solids and to some extend evaporation and emulsification. These behaviours are crucial for the realism of the slick, we make sure that they are computed at each time step (or once every several time steps for more minute changes of evaporation, dispersion and emulsification).

This means that at a time t, the volume V_t, density ρ_t and viscosity μ_t of the slick are:

$$\begin{aligned}
V_t &= V_\theta(1 - E_t + Y_t) \\
\rho_t &= (\rho_{oil} + C_3(E_t))(1 - Y_t) + \rho_{water}(Y_t) \\
\mu_t &= \mu_{oil} \times exp\left[C_4 E_t\right] \times exp\left[\frac{2.5Y_t}{(1 - C_3 Y_t)}\right]
\end{aligned}$$

The velocity of the oil from spreading and advection are covered in Section 4. The calculation of derivatives for these quantities are covered in Appendix C.

Considered at start-up: these phenomena evolve to slowly to be modelled in real-time, this includes photo-oxidation, sedimentation, and biodegradation. As mentioned, they affect the oil over a long span of time, so we can specify the age of the slick at start-up, and apply their changes on the oil.

Not considered: these are the behaviours that are unnoticeable; dissolution, that happens almost immediately after contact with water, and dispersion, which is for the most part negligible.

4 Our Approach: The Shallow Water Equations

VSTEP has a running simulator for boats (Nautis3), comprising a procedural sea generator (Section 2.1) for the scenery, and SWE for boat/water interaction. As their simulation of water is still in development, they only implement a simple version of the Shallow Water Equations (SWE) (Section 4.2.1), which is a simplified model made for (serious) games, and which is based on the Navier-Stokes Equations (NSE)(Section 3.1.1).

In this section, we define the input and outputs we have to take into account for our simulation. We then describe a simple SWE solver for a better understanding of its mathematical representation of water. This solver is then used in Section 5 as a base for the oil and water simulation.

4.1 Inputs and Variables

We work with grids of $n \times m$ cells, where the cells have edge lengths of $x \times y$ respectively. We follow the evolution of the simulation over time t, and dt refers to the elapsed time between each time-step.

Inputs

- **Water**: we store the water on a 2D grid, where each cell center stores the height of the water η_{water}, the ground elevation at this point g_{water} and the total height h_{water}. At the edges of the cells, we store the horizontal velocity $\vec{U}_{water}$ (figure 9).
- **Oil Slick**: the oil slick is stored on a grid of the same format as the water, storing η_{oil}, g_{oil}, h_{oil} and $\vec{U}_{oil}$ at the same locations as its water equivalent. We also store global variables for the slick, which are the slick density ρ_{oil}, viscosity μ_{oil}, evaporated fraction E_{oil} , water content fraction Y_{oil}
- **Wind**: it is a vector field $\vec{V}_w$, taking effect in the whole simulation.
- **Current**: There are no currents present in the simulation other than the one generated by the movement of the fluids. The currents are thus described solely by the fluid's horizontal velocities $\vec{U}_{water}$ and $\vec{U}_{oil}$.
- **Boats**: the vessels influence the water by direct contact (influencing the water height). The boats propulsion system adds a repulsive force at the rear of the ship, pushing water and oil away (and the ship forward). This influence by the propellers is directly added to the relevant cells $\vec{U}_{water}$ and $\vec{U}_{oil}$.

4.2 Shallow Water Simulation

We next explain how we simulate water on a large scale.

Oil, as any fluid, can be modelled using the Navier-Stokes Equation (Section 3.1.1), however, it is notoriously difficult to discretize and simulate exactly. Therefore, we approximate it by the Shallow Water Equation SWE (Section 4.2.1). The SWE reduces the complexity of the equation by representing the fluids as a two-dimensional height field.

We use the following notation (as in [MSJT08]):

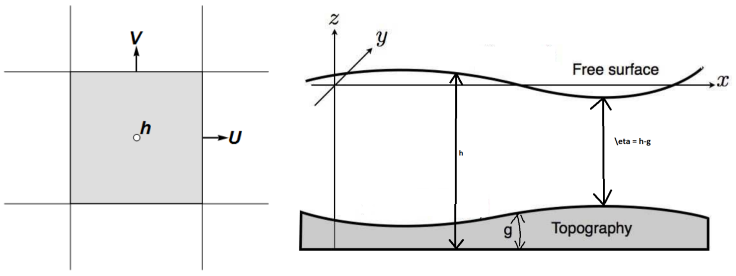


Figure 9: Values and their meaning in SWE

- h denotes the height of the fluid above a fixed level which is "zero".
- g is the height of the ground below the fluid (above zero-level).
- η denotes the height of the fluid above ground, $\eta = h - g$.
- $\vec{U} = (\vec{u}, \vec{v})^\top$ denotes the velocity of the fluid in the horizontal plane.

The above allows us to represent the state of the fluid as follow:

$$S_t = S\left(x,y,t\right) = \begin{pmatrix} \eta\left(x,y,t\right) \\ \vec{u}\left(x,y,t\right) \\ \vec{v}\left(x,y,t\right) \end{pmatrix} = \begin{pmatrix} \eta_t \\ \vec{u}_t \\ \vec{v}_t \end{pmatrix} \tag{9}$$

Where the fluid state S at time t is represented by its fluid column η, and its horizontal velocity $\vec{u}$.

4.2.1 Shallow water equations

The shallow-water equations (SWE) is an approximation of the Navier-Stokes equations, that reduce the problem of a three-dimensional fluid motion to a two-dimensional description, using a height-field representation. To reduce the complexity, several assumptions are made:

- The fluid pressure is *hydrostatic*, meaning that the vertical velocity is constant over time and equal 0;
- The fluid is *incompressible*, meaning that the density is uniform over space;
- The viscosity of the fluids is negligible.

Several works [ZB11] handle density changes over space, and other complex multi-fluids set-ups. However we consider the oil as non-miscible, as a distinctive layer above water, and where density and viscosity change uniformly over time.

Considering the above rules, the basic formulation of SWE can be written as follow:

$$\frac{\delta\eta}{\delta t} + (\nabla\eta)\vec{u} = -\eta\nabla \cdot \vec{u} \tag{10}$$

$$\frac{\delta\vec{u}}{\delta t} + (\nabla\vec{u})\vec{u} = a_n \nabla h \tag{11}$$

With a_n as the vertical acceleration of the fluid (e.g., gravity).

4.3 Discretization of SWE

We next use the shallow-water equations to model the base algorithm proposed by [TMFSG07]. We then modify this basic SWE solver, and add modifications to consider floating oil with changing properties over time.

4.3.1 Conservative vs non-conservative form

Before laying out an algorithm for the SWE, we need to choose if we go for a conservative or non-conservative form of discretization. This affects the difficulty (and consequently, the efficiency) of the calculation, as well as its precision [LeV02].

Conservative and non-conservative forms of equations are directly related to the definition of their derivatives. We consider the following example of a derivation, using arbitrary variables λ and ω:

$$\frac{\delta \omega \lambda}{\delta x}$$

A classical discretization in the conservative form is:

$$\frac{\delta \omega \lambda}{\delta x} \approx \frac{(\rho\lambda)_i - (\omega\lambda)_{i-1}}{\Delta x}$$

whereas in the non-conservative form we split the derivative apart, obtaining the following discretization:

$$\omega \frac{\delta \lambda}{\delta x} + \lambda \frac{\delta \omega}{\delta x} = \omega_i \frac{\lambda_i - \lambda_{i-1}}{\Delta x} + \lambda_i \frac{\omega_i - \omega_{i-1}}{\Delta x}$$

While these two discretization converge to the same in the limit, they behave differently otherwise. This is mostly evident when we expand it on an grid. Consider a one-dimensional grid of 4 points, where the boundary vertices are $i = 0$ and $i = 3$. Both equations expand as follow:

$$\frac{(\omega\lambda)_1 - (\omega\lambda)_0}{\Delta x} + \frac{(\omega\lambda)_2 - (\omega\lambda)_1}{\Delta x} + \frac{(\omega\lambda)_3 - (\omega\lambda)_2}{\Delta x} \tag{α}$$

$$\omega_1 \frac{\lambda_1 - \lambda_0}{\Delta x} + \lambda_1 \frac{\omega_1 - \omega_0}{\Delta x} + \omega_2 \frac{\lambda_2 - \lambda_1}{\Delta x} + \lambda_2 \frac{\omega_2 - \omega_1}{\Delta x} + \omega_3 \frac{\lambda_3 - \lambda_2}{\Delta x} + \lambda_3 \frac{\omega_3 - \omega_2}{\Delta x} \tag{β}$$

For the expanded conservative form (α), we can see that, when summing the derivatives, we end up only with the boundary terms ($i = 0$ & $i = 3$) as the interior points have cancelled out. This highlight that the conservative form is only dependent on the boundary conditions. It also means that for what goes in the simulation must comes out, we have *conservation* of values over time (i.e. a closed differential form).

However for the non conservative form (β), cancellation is not possible, thus adding grid point also makes the number of terms grow. Simply put, what comes in *does not* balance what goes out, hence the name *non-conservative.*

Note that non-conservative equations introduce, by the way they are calculated, some errors, that is viewed as artificial viscosity in the case of fluid simulation (slower propagation of the fluid). This makes them not suitable to calculate, in fluids, hydraulic jump and other abrupt discontinuities known as shocks (occurring at supersonic speeds), were the computation is likely to crash or yield unrealistic results.

Conservative equations do not introduce those errors and hold for more complex scenarios, but they come at a much higher relative cost (more calculations steps for their resolution). Thus for our SWE solver, using the non conservative form is perfectly suitable, as the oil does, in the general case, expand at a subsonic rate, and it is also a viscous fluid, so the introduction of artificial viscosity is not a problem (e.g. [MSJT08] uses the non-conservative form).

4.3.2 Choice of grid

This method makes the use of a staggered grid, instead of naive collocated grids.

Collocated grids are the standard grid type, were all types of values are stored per cells (usually at their center), accessing these values at position (x, y) would simply resume to quering the grid a position (x, y).

A staggered grid is a different setting for the spatial discretization, where some variables are set on the vertices [HW65, Lil61]. As seen in Figure 9, our staggered grid stores the different heights of the column in the center of each cells (h, g and η for oil and water), and the velocities $\vec{u}$ and $\vec{v}$ on the edges.

Querying heights values stays the same as with the collocated grid, and querying the velocities at the same spatial position (x, y) becomes a linear interpolation:

$$\vec{u}_{(x,y)} = (\vec{u}_{(x+1/2,y)} + \vec{u}_{(x-1/2,y)}) * \frac{1}{2}$$

or

$$\vec{v}_{(x,y)} = (\vec{v}_{(x,y+1/2)} + \vec{v}_{(x,y-1/2)}) * \frac{1}{2}$$

Although seemingly tedious, staggered grids avoids extra calculations. As non staggered grids usually require error correction at each steps (to avoid the "odd-even decoupling problem", leading to the appearance of checkerboard patterns and other instabilities). The staggered grid requires no correction and is particularly fit (as it was developed) for our computing method (and other fluid/gases simulations).

4.3.3 Simplistic non-conservative discretization

We follow the techniques proposed originally by [Sta99] and applied to a basic SWE solver in [TMFSG07] and [Mol11]. This method is called semi-Lagrangian, despite being nonetheless Eulerian, as we calculate the fluid position over the next time step by following imaginary arbitrary particles. The solver follow the mathematical Lax-Wendroff method to calculate the values at each time step. More precisely, a variant named the MacCormack method, as it is widely used in fluid simulation for its simplicity of understanding and implementing.

The basic algorithm is:

Algorithm 1 The full shallow-water time step

```
function SHALLOW-WATER-STEP( )
    Resolve_Boundaries()
    $h' = \text{Advect\_Water}(\eta, \vec{U}, g)$
    $\vec{U}' = \text{Advect\_Velocities}(h, \vec{U}, g)$
    $\eta$ and $h = \text{Update\_Height}(\vec{U}')$
    $\vec{U} = \text{Update\_Velocities}(\vec{U}', h')$
```

A full time step following the MacCormack method is decoupled into two interleaving steps. The advection (also named "half-time step" or "predictor step") and the update step ("full-time step" or "corrector step"). As seen, there is a need to store temporary values for h' and $\vec{u}'$, as the half time step values in the x and y direction are at imaginary positions (storing 6 temporary values per cell).

Advection step: water height and velocities advection are computed as follow:

Algorithm 2 water height advection on both axis

```
function ADVECT_WATER($\eta, \vec{U}, g$)
    for $i = 1$ to $n$ do
        for $j = 1$ to $n - 1$ do
            $h_{x(i,j)} = \eta_{(i+1/2,j+1)} - \vec{u}_{(i+1/2,j+1)} * \frac{\Delta t}{\Delta x} + g_{(i+1/2,j+1)}$
    for $i = 1$ to $n - 1$ do
        for $j = 1$ to $n$ do
            $h_{y(i,j)} = \eta_{(i+1,j+1/2)} - \vec{v}_{(i+1,j+1/2)} * \frac{\Delta t}{\Delta x} + g_{(i+1,j+1/2)}$
    return $h'$
```

Algorithm 3 water velocity advection on both axis

```
function ADVECT_VELOCITIES($h, \vec{U}, g$)
    Considering $A(a,b)_{(i,j)} = \frac{a(i,j)*b(i,j)}{h(i,j)}$
    for $i = 1$ to $n$ do
        for $j = 1$ to $n - 1$ do
            $\vec{u}_{x(i,j)} = \vec{u}_{(i+1/2,j+1)}$
                $-[[A(\vec{u},\vec{u})_{(i+1,j+1)} + \frac{g}{2h(i+1,j+1)^2}]$
                $-[A(\vec{u},\vec{u})_{(i,j+1)} + \frac{g}{2h(i,j+1)^2}]] * \frac{\Delta t}{2\Delta x}$
            $\vec{v}_{x(i,j)} = \vec{v}_{(i+1/2,j+1)}$
                $-[A(\vec{u},\vec{v})_{(i+1,j+1)} - A(\vec{u},\vec{v})_{(i,j+1)}]\frac{\Delta t}{2\Delta x}$
    for $i = 1$ to $n - 1$ do
        for $j = 1$ to $n$ do
            $\vec{u}_{y(i,j)} = \vec{u}_{(i+1,j+1/2)}$
                $-[A(\vec{u},\vec{v})_{(i+1,j+1)} - A(\vec{u},\vec{v})_{(i+1,j)}]\frac{\Delta t}{2\Delta x}$
            $\vec{v}_{y(i,j)} = \vec{v}_{(i+1,j+1/2)}$
                $-[[A(\vec{v},\vec{v})_{i+1,j+1} + \frac{g}{2h(i+1,j+1)^2}]$
                $-[A(\vec{v},\vec{v})_{i+1,j)} + \frac{g}{2h(i+1,j)^2}]] * \frac{\Delta t}{2\Delta x}$
    return $\vec{U}'$
```

The purpose of the Advection step is to compute an imaginary particle at the previous time step (Algorithm 2 and 3). Despite being fairly tedious, it is very simple: we take the point at an imaginary position X (water height or velocity, at either the center or the edge of our cells), and calculate its position at a previous time step x' along the u or v direction (note that X is at a well defined position on the grid, but it is not the case for x').

For the height of the water surface at the (imaginary) particle position (Algorithm 2), h' is simply $(\eta - \vec{U} + g)$ as $h = \eta + g$ (we consider the floor static).

As for the estimated velocity (Algorithm 3), $\vec{u}'$ (or $\vec{v}'$) is the current velocity minus an estimated neighbouring velocity attenuated by the height of the water column and gravity (like in reality, with less depth, waves tend to accelerate and diffuse less).

Update step water height and velocities update step :

Algorithm 4 water height update

```
function UPDATE_HEIGHT($\vec{U}'$)
    for $i = 1$ to $n$ do
        for $j = 1$ to $n - 1$ do
            $\eta(i,j) \mathrel{-}= [\vec{u}_{x(i-1/2,j-1)} * \frac{\Delta t}{\Delta x} - \vec{v}_{y(i-1,j-1/2)} * \frac{\Delta t}{\Delta y}$
            $h(i,j) = \eta(i,j) + g(i,j)$
    return $\eta, h$
```

Algorithm 5 water velocity update

```
function UPDATE_VELOCITIES($\vec{U}', h'$)
    Considering $A_x(a,b)_{(i,j)} = \frac{a_x(i,j) * b_x(i,j)}{h_x(i,j)}$
    for $i = 1$ to $n$ do
        for $j = 1$ to $n - 1$ do
            $\vec{u}(i,j) \mathrel{-}= [[A_x(\vec{u},\vec{u})_{(i,j-1)} + \frac{g}{2 * h^2_{x(i,j-1)}}]$
                $-[A_x(\vec{u},\vec{u})_{(i-1,j-1)} + \frac{g}{2 * h^2_{x(i-1,j-1)}}]] * \frac{\Delta t}{\Delta x}$
                $-[A_y(\vec{u},\vec{v})_{(i,j-1)} - A_y(\vec{u},\vec{v})_{(i-1,j-1)}] * \frac{\Delta t}{\Delta y}$
            $\vec{v}(i,j) \mathrel{-}= [[A_y(\vec{v},\vec{v})_{(i-1,j)} + \frac{g}{2 * h^2_{y(i-1,j)}}]$
                $-[A_y(\vec{v},\vec{v})_{(i-1,j-1)} + \frac{g}{2 * h^2_{y(i-1,j-1)}}]] * \frac{\Delta t}{\Delta y}$
                $-[A_x(\vec{u},\vec{v})_{(i-1,j)} - A_x(\vec{u},\vec{v})_{(i-1,j-1)}] * \frac{\Delta t}{\Delta x}$
    return $\vec{U}$
```

The correction step (Algorithm 4 and 5) is there to obtain the correct values for rendering and the next time step.

For the height of the fluid column and the total height (Algorithm 4), it is simply $\eta_{(t+1)} = \eta_{(t)} - \vec{u}'$ and $h = \eta - g$.

As for the final velocities (Algorithm 5), it is where the half time step advected values become visibly useful. The new velocity for u is $u_{(t+1)} = \vec{u}_{(t)} - [\frac{\vec{u}'^2}{h'} + \frac{g}{2h'^2}] - [\frac{\vec{u}' * \vec{v}}{h'}]$ and same goes for $\vec{v}$ with some adjustments [MSJT08].

We note that in the above algorithms, many of the variables are located at a half spatial step (e.g. $\vec{u}_{(i+1/2,j+1/2}$); this simply suggest that a basic linear interpolation to obtain their values is necessary (e.g. $[\vec{u}_{(i,j)} + \vec{u}_{(i+1,j+1)}]/2$).

4.3.4 Boundary behavior

We next define the behavior of the boundary, in order for the simulation to be coherent.

For the water layer, we cannot hope to compute the whole map using SWE (as VSTEP training worlds are up to 200 km^2). The solution is to narrow our focus to important areas (where the oil is), and the rest of the ocean is then computed following VSTEP current model (procedural water). This is done by having absorbing boundaries at the far limits of our grid (at the edges of the map), in order to avoid reflection of the wave. Absorbing boundaries here means wave energy absorbtion, so no volume is lost over time.

In the case where grid boundaries are against a wall or other reflective structure, we want reflective boundary conditions. The two boundary conditions are computed as follow (here only the left side of the grid is considered):

Reflective boundary Conditions — Absorbing boundary conditions

$$\begin{aligned} \eta_{(0,j)} &= \eta_{(1,j)} \\ \vec{u}_{(0,j)} &= 0 \\ \vec{v}_{(0,j)} &= 0 \end{aligned} \tag{12}$$

$$\begin{aligned} \eta_{(0,j)} &= \eta_{(1,j)} \\ \vec{u}_{(0,j)} &= \vec{u}_{(1,j)} \\ \vec{v}_{(0,j)} &= \vec{v}_{(1,j)} \end{aligned} \tag{13}$$

4.3.5 Flag system

To reduce costs, we avoid computing changes on areas without fluids. A flag system is thus put in place. Every cells contains contains a flag f that determines if the cell is updated or not. The flag system for a basic SWE is presented in [MSJT08]. The flags are computed per cells at each time steps as well as a ratio r. This ratio r keeps track of the "wetness" of the cell, with a value bounded between 0 (for dry, i.e devoid of the tracked fluid) and 1 (fully immersed). Any value in between indicates how much of the cell is covered by the fluid (e.g. $r = 0,5$ would represent an half immersed cell). The computation of the flags is made in two steps as follows:

Algorithm 6 Calculate the needed values to update the flags

```
function PRECOMPUTE_FLAGS
    for $i = 1$ to $n$ do
        for $j = 1$ to $n$ do
            $g_{min}(i,j) = (g(i,j) + \min g(\mathbf{p})) * \frac{1}{2}$
            $g_{max}(i,j) = (g(i,j) + \max g(\mathbf{p})) * \frac{1}{2} + \epsilon_H$
            $\eta_{max}(i,j) = (\eta(i,j) + \max \eta(\mathbf{p})) * \frac{1}{2}$
```

With $\mathbf{p}$ the four direct neighbour of our cell (i,j). The first step computes the maximum and minimum ground height of each cells (g_{min} and g_{max}) and the maximum fluid thickness (η_{max}). The addition of the small value ϵ_H is to prevent g_{min} to be equal to g_{max} in flat areas, we choose epsilon to be $\epsilon_H = 0.000001$. We then update the flags for each cells:

Algorithm 7 Flag computation

```
function UPDATE_FLAGS
    for $i = 1$ to $n$ do
        for $j = 1$ to $n$ do
            if $(g(i,j) \leq g_{min}(i,j))$ && $(\eta_{max}(i,j) < \epsilon_{\eta_{max}})$ then
                $f(i,j) = DRY$
                $r(i,j) = 0$
            else if $g(i,j) > g_{max}$ then
                $f(i,j) = FLUID$
                $r(i,j) = 0$
            else
                $f(i,j) = DRY$
                $r(i,j) = (g(i,j) - g_{min}(i,j))/(g_{max}(i,j) - g_{min}(i,j))$
```

Every cells is marked with a flag stating either DRY or $FLUID$, and the fill rate r is stored. There is also an other small value $\epsilon_{\eta_{max}}$ (equal to 0.00001 in our implementation) that prevents the water flowing to other cells if the fluid layer is very thin; this is interpreted as an artificial viscosity.

It is important to note that this flag system was implemented for an early prototype and is not used as is in the NAUTIS simulator. Instead, to fit their expanding grid system, a cell that would be marked as DRY is automatically removed from the grid. For more details refer to section 7.

5 Addition of a second layer of fluid

So far, we reviewed how to properly model a single fluid using the SWE efficiently using the proper discretization, staggered grids and flags. We next review how to add an extra layer.

To add a second layer, we modify the SWE algorithm to work both for water and for oil (one copy to resolve water, one to resolve oil). We also modify the SWE Algorithms to reproduce oil and water behaviours.

5.1 SWE Modifications for Oil

SWE stores density and viscosity with the variables ρ and μ, respectively (see Section 3.3). For the water layer, density ρ and viscosity μ are both equal to 1. For the oil, we update ρ and μ in a slow pace (every minute, or 3000 steps), although their change will be unnoticeable on short term.

We also consider the changes of oil volume over time (due to water incorporation), despite them being minute. For these changes, we adjust the oil volume per cells over time by $1 + \%_{increased}$ before computing the spreading (also at large interval, every minute).

The details for the computations of derivatives for volume density and viscosity are in Appendix C.

5.2 The Algorithm

To adapt the algorithm for an oil layer above the water, we have to consider the changes necessary to account for the interacting forces between oil and water (enforcing floating oil and account for the oil-water pressure), as well as how they integrate into the existing algorithm.

First, we enforce the oil to be floating above the water at all time. This means that the oil "ground" level is equal to the surface of the water layer at all time, i.e. $g_{oil} = h_{water}$. This can be done easily using pointers, if provided by the programming language.

Next, we model the forces applied by the oil on the water, and vice versa. The water has to account for a downward force exerted by the oil (pressure from oil movement and weight of the oil), and the oil accounts for an upward force from water displacement. We model the pressure of oil on water after the Bernoulli equation:

$$\frac{v^2}{2} + gz + \frac{p}{\rho} = constant \tag{14}$$

With v the fluid flow speed, g the gravity constant, z the elevation, p the pressure and ρ the density of the fluid. This equation is normally used to relate fluids velocity with pressure, the higher the velocity of a fluid, the less pressure: it exerts on its surroundings.

We can modify Equation 14 to extract the static pressure SP_{water} (Equation 15, a downward force from the oil column, always positive) and the dynamic pressure DP_{water} (Equation 16) of the oil on the water (up- or downward force resulting of the movement of the oil). Static pressure refers to the pressure exerted by the fluid if considered at rest (the weight of the fluid), whereas dynamic pressure refers to the pressure exerted by the movement of the fluid (its kinetic energy).

$$SP_{water} = V_{oil} \cdot \mu \cdot g \tag{15}$$

$$DP_{water} = \mu_{oil} \cdot \vec{v}^2_{oil} \cdot \frac{1}{2dt} \cdot \alpha \tag{16}$$

With V_{oil} the oil volume, $\vec{v}$ the instant velocity of the fluid. α is the direction of the fluid column ($\alpha = -1$ if the column is losing volume at time t and $\alpha = 1$ if it is gaining volume).

The oil is also influenced by the water movement, but only by the dynamic pressure DP_{oil} (as the water is under the oil):

$$DP_{oil} = \mu_{water} \cdot |\vec{v}|^2_{water} \cdot \frac{1}{2dt} \cdot -\alpha \tag{17}$$

We note that normally pressure is a scalar quantity, i.e. pressure apply in every direction. Here the terms static and dynamic pressure are referring to the approximate global force vector applied by the oil on the water and vice versa.

We integrate these pressures into the SWE algorithm, by defining the Static_Pressure() and Dynamic_Pressure() functions (Algorithm 8, 9). The function signum() in algorithm 8 is the "sign of" function, returning 1 for positive arguments and -1 for negatives.

Algorithm 8 Dynamic Pressure

```
function DYNAMIC_PRESSURE($\Delta\eta, \eta, \rho$)
    return signum $(\Delta\eta) * \eta * \rho * \Delta\eta^2 * \frac{1}{2dt}$
```

Algorithm 9 Static Pressure

```
function STATIC_PRESSURE($\eta, \rho$)
    return $\frac{g}{2} * \eta^2 * \rho$
```

The parameters of these functions are the instant vertical velocity of the fluid $\Delta\eta$, the volume of fluid in the column η, and the density of the fluid ρ (lighter fluid exert less pressure). They both return a force that is then applied to the fluids.

We also modify the SWE algorithms to include Static_Pressure() and Dynamic_Pressure() for both fluids. When computing the pressure, there is a need for the instant vertical velocity $\Delta\eta$ of the fluid, we thus store it at the half time step under $\Delta\eta_x$ and $\Delta\eta_y$ (see Algorithm 10) and at full time step under $\Delta\eta$ (Algorithm 11).

Algorithm 10 Height Advection for 2 layers SWE

```
function ADVECT_HEIGHT_CORRECTED($\eta, \vec{U}, g$)
    for $i = 1$ to $n$ do
        for $j = 1$ to $n - 1$ do
            $\Delta\eta_{x(i,j)} = -\vec{u}_{(i+1/2,j+1)} * \frac{\Delta t}{\Delta x}$
            $h_{x(i,j)} = \eta_{(i+1/2,j+1)} + \Delta\eta_{x(i,j)} + g_{(i+1/2,j+1)}$
    for $i = 1$ to $n - 1$ do
        for $j = 1$ to $n$ do
            $\Delta\eta_{y(i,j)} = -\vec{v}_{(i+1,j+1/2)} * \frac{\Delta t}{\Delta x}$
            $h_{y(i,j)} = \eta_{(i+1,j+1/2)} + \Delta\eta_{y(i,j)} + g_{(i+1,j+1/2)}$
    return $h'$
```

Algorithm 11 Height Update for 2 layers SWE

```
function UPDATE_HEIGHT_CORRECTED($\vec{U}'$)
    for $i = 1$ to $n$ do
        for $j = 1$ to $n - 1$ do
            $\Delta\eta_{(i,j)} = -[\vec{u}_{x(i-1/2,j-1)} * \frac{\Delta t}{\Delta x} - \vec{v}_{y(i-1,j-1/2)} * \frac{\Delta t}{\Delta y}$
            $\eta(i,j) \mathrel{+}= \Delta\eta_{(i,j)}$
            $h(i,j) = \eta(i,j) + g(i,j)$
    return $\eta, h$
```

Once we have the instant velocities, we can modify the rest of the algorithm to account for the different pressures. This comes by modifying the A function in the half and full time step (from Algorithm 3 and 5).

For oil it becomes:

$$\begin{aligned}\mathbf{A}(a,b)_{(i,j)} &= \frac{a(i,j) * b(i,j)}{h(i,j)} \\ &\quad - \text{Dynamic\_Pressure}(water \to \Delta\eta_{(i,j)}, water \to \eta_{(i,j)}, water \to \rho)\end{aligned} \tag{18}$$

$$\begin{aligned}\mathbf{A_x}(a,b)_{(i,j)} &= \frac{a_x(i,j) * b_x(i,j)}{h_x(i,j)} \\ &\quad - \text{Dynamic\_Pressure}(water \to \Delta\eta_{x(i,j)}, water \to \eta_{x(i,j)}, water \to \rho)\end{aligned} \tag{19}$$

And for Water:

$$\begin{aligned}\mathbf{A}(a,b)_{(i,j)} &= \frac{a(i,j) * b(i,j)}{h(i,j)} \\ &\quad + \text{Dynamic\_Pressure}(oil \to \Delta\eta_{(i,j)}, oil \to \eta_{(i,j)}, oil \to \rho) \\ &\quad + \text{Static\_Pressure}(oil \to \eta_{(i,j)}, oil \to \rho)\end{aligned} \tag{20}$$

$$\begin{aligned}\mathbf{A_x}(a,b)_{(i,j)} &= \frac{a_x(i,j) * b_x(i,j)}{h_x(i,j)} \\ &\quad + \text{Dynamic\_Pressure}(oil \to \Delta\eta_{x(i,j)}, oil \to \eta_{x(i,j)}, oil \to \rho) \\ &\quad + \text{Static\_Pressure}(oil \to \eta_{x(i,j)}, oil \to \rho)\end{aligned} \tag{21}$$

With these new version of the A function, we can now execute the full algorithm twice in a row (once for each layer) to calculate a full timestep (Algorithm 12).

Algorithm 12 The full Shallow Water time step for 2 fluids

```
function SHALLOW_WATER_STEP_CORRECTED( )
    Resolve_Boundaries ()
    Water Resolution
    $h'_{water}$ = Advect_Height_Corrected($\eta, \vec{U}, g$)
    $\vec{U}'_{water}$ = Advect_Velocities_Water($h, \vec{U}, g$)
    $\eta_{water}$ and $h_{water}$ = Update_Height_Corrected($\vec{U}'$)
    $\vec{U}_{water}$ = Update_Velocities_Water($\vec{U}', h'$)
    Oil Resolution
    $h'_{oil}$ = Advect_Height_Corrected($\eta, \vec{U}, g$)
    $\vec{U}'_{oil}$ = Advect_Velocities_Oil($h, \vec{U}, g$)
    $\eta_{oil}$ and $h_{oil}$ = Update_Height_Corrected($\vec{U}'$)
    $\vec{U}_{oil}$ = Update_Velocities_Oil($\vec{U}', h'$)
```

6 Interaction with solids

6.1 Floating objects

Immersed objects are subject to buoyancy, an upward force making light objects float. Buoyancy is expressed as follows:

$$F_b = \rho g V \tag{22}$$

Where F_b is the buoyant upward force, ρ the fluid density, g gravity, and V the submerged volume. This mean that objects that are denser than the fluid sink, and the less dense ones are partially submerged.

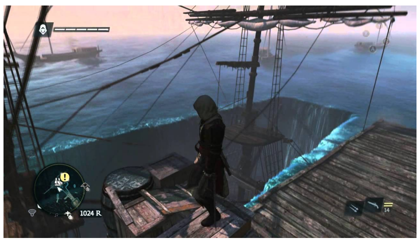
As the basic SWE algorithm only keeps track of the elevation h of the water columns, adding dense sinking objects would create holes in the water (see figure 10a, sunken ships problem) and would prevent circulation of fluids over the solid. We directly account for this, and use insights for [GDC08b] and the work of [OH95, Gom00, Tes04]. Their approach is to let the solid pass through the fluid, and add the solid submerged volume as fluid in the simulation. By adding the submerged volume to the solution, we indirectly generate the waves from the solid-fluid interaction as the added fluid volume propagates to the other cells of the simulation (see Figure 10b). The resulting equation for buoyancy becomes:

$$F_b = -\Delta u \cdot \rho g \cdot dxdy \tag{23}$$

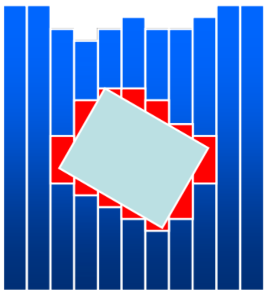
Where $dxdy$ is the surface of the cell, and Δu is the volume of fluid replaced by the body. Note that the force is applied at the center of the cells, and can be summed at the center of gravity of our solid (resulting in a not always upward force due to different buoyant forces).

Another aspect to be considered is the reflection of waves on partially submerged solids. If a wave crashes in a floating solid (boat, buoy), we reflect it in the same manner we reflect waves at boundary conditions. That is, for water from a free cell denoted as $free$ and the "occupied" cell obj :

$$\begin{aligned} \eta_{obj} &= \eta_{free} \\ \vec{u}_{obj} &= 0 \\ \vec{v}_{obj} &= 0 \end{aligned} \tag{24}$$



(a)



(b)

Figure 10: Submerged boat causing the Moses effect in "Assassin's creed black flag" (10a), who uses the same SWE system for its water, and a visual representation of how to solve it, by accounting for the displaced water volume(10b)

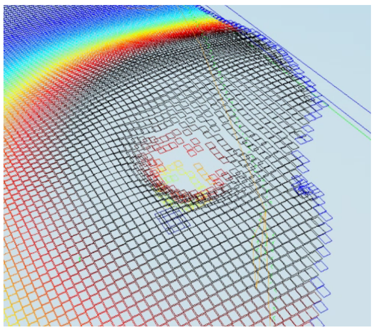
6.2 Addition and deletion of fluids

Another possible interaction of objects with fluids comes from an the object forcing displacement (i.e propeller pushing the oil), acting as a water source (i.e leaking pipe) or as a drain (i.e removal via a pump). Those three interactions are handled in the same function Add_Delta() (algorithm 13), handling the changes between 2 updates:

Algorithm 13 Add Delta

```
function ADD_DELTA($\vec{W}$, $\vec{O}$)
    Resolve_Boundaries ()
    $\vec{u}_{water} \mathrel{+}= \vec{W}.x$
    $\vec{v}_{water} \mathrel{+}= \vec{W}.y$
    $h_{water} \mathrel{+}= \vec{W}.z$
    $\vec{u}_{oil} \mathrel{+}= \vec{O}.x$
    $\vec{v}_{oil} \mathrel{+}= \vec{O}.y$
    $h_{oil} \mathrel{+}= \vec{O}.z$
```

Where $\vec{W}$ and $\vec{O}$ the changes over one time step in velocity (x and y component) and height (z component). We show examples in Figure 11, where, on a test oil slick, we delete and push oil away at each time step by applying a pushing or drainning vector ($\vec{O} = (0, 0, -10)$ for a left push and $\vec{O} = (0, 10, 0)$ to drain the oil).


(a)

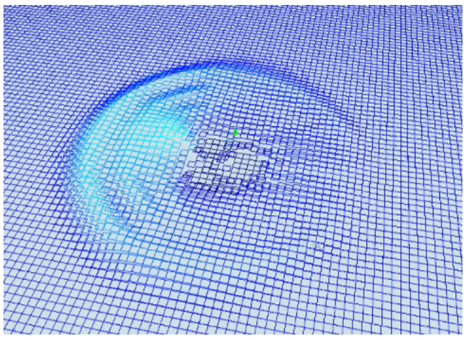

(b)

Figure 11: Two simulation instances of an oil slick. Left: Add_Delta() deleting oil. Right: Add_Delta() forcing oil away to left, creating ripples in one direction. Color represent relative thickness of the slick, where a thin layer is blue and a thick layer black.

7 Fluid Boundary Expansion

In addition to the flag system made to optimize computations, we expand and reduce the grid size to fit the active water and oil active cells. This means that cells of the grid are added/deleted at runtime whenever needed. A cell is considered active if there is water movement ($|\Delta\eta|$ or $|\Delta\vec{u}|$ is above a certain threshold, here 0.02), or if there is oil on it. If a cell does not meet this requirement, we take it out of the grid and continue the simulation.

This is possible by the way the NAUTIS system implements the SWE equations, using 2D double-linked list, making it easy for every cells to access their neighbours (e.g. `cell.right.value`), but also to delete the edges of the grid (e.g. a cell is deleted and its left neighbour updates to `cell.right = NULL`).

For cells at the edges, the missing neighbours are replaced with pointers to a "dummy" default cells that contains default values (default water height, no oil and no velocities).Figure 12 illustrate this by showing a cell $x_{(m,n)}$, its existing neigbours $x_{(m,n+1)}$ and $x_{(m,n-1)}$ and its dummy neighbour $x_{(m+1,n)}$. As we always expand the grid when water and oil nearly reaches the borders, only a negligible amount of water is lost over time.

It is to note that the base algorithm was already present in the NAUTIS software and iterated upon. An error appears while querying the dummy cells in certain cases and we discuss it in section 9.2.

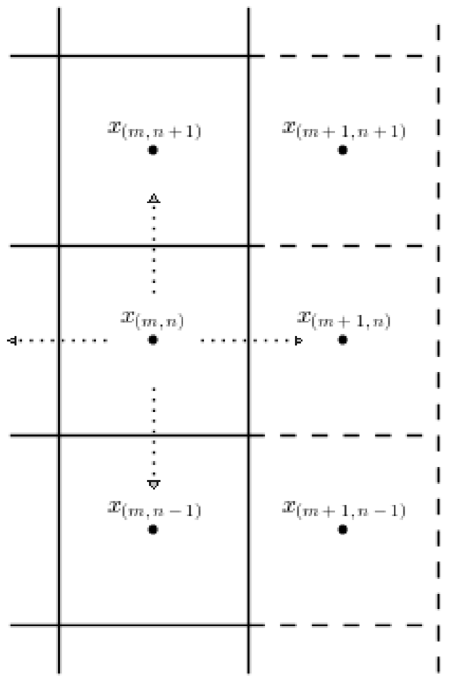


Figure 12: Example of the grid cells,during a time step: $x_{(m,n)}$ needs access to all its direct neighbor. As a border cell, $x_{(m+1,n)}$ does not exit, $x_{(m,n)}$ thus uses default values with no oil, water at sea level and no velocites

8 Performance improvement

We improve the already decent performance of SWE using Streaming SIMD Extensions (SSE). Essentially, we replace all the calculations done on single *float* values (height, velocity etc ...) by the $\_\_m128$ data type (or $\_\_m256$ or $\_\_m512$ on appropriate processors). Using $\_\_m128$ permits to work with 4 float values at once. For example, calculating the average of 2 cell water heights is done as follows:

```
//Assuming cell.height and cell.rightNeighbour.height
//are arrays of size N.

__m128 *cellHeight, *nghbrCellHeight, half, result[(N%4)+1];

cellHeight      = (__m128d*)cell.height;
nghbrCellHeight = (__m128d*)cell.rightNeighbour.height;
half        = _mm_set1_ps(0.5f)         // used to divide by 2

for (i = 0; i < numberOfCells/4; i++){
   //respectively add the heights and multiply by 0.5
   result[i]   = _mm_add_ps (cellHeight[i],
                       nghbrCellHeight[i]);
   result[i]   = _mm_mul_ps (result , half);
}
```

We thus use it to compute the half and full time-step of 4 square neighbour cells in parallel. If there is no 4 cells to compute (edges of the grid) we fill the missing values with ”dummy” variables (zeroes) that are discarded at the end of the computation.

9 Results

9.1 Simulations results

We implemented the algorithm in the NAUTIS simulator as an interchangeable fluid system with their already existing SWE simulating their water environment.

Visuals of several test scenarios can be found in figure 13 to 15. These scenarios includes basic spreading of oil with different viscosities, addition, deletion, and pushing of oil. There are also specific cases such the visualisation of how water is affected by the oil or how the oil interacts with boundaries.

For each of these simulations, the computation time depends on the grid size and not on what is happening within the simulation. While testing, for a fixed grid size (64 by 64 cells), the average computation time for 30 time steps (equivalent to 1 second) is around 10 ms without SSE and around 3 ms with SSE. Limits of the simulations are reached for grids of more than 3500 by 3500 cells, where the computations with SSE is more than half a second per time steps, but for the average slick, the computation time using SSE stays correct, averaging 20 ms per time steps, or 50 fps. Results are found in table 1.

Grid size	64 by 64	Average slick	>3500x3500
With SSE	$0.1ms$	$\sim 20ms$	$500ms$
Without SSE	$0.3ms$	$\sim 50ms$	$>1s$

Table 1: Average computation time per time steps (in ms), with and without SSE. The 64 by 64 Slick is a bounded environment for testing, and the average slick is comparable to the slicks seen in figures 13 to 15 ($\sim$ 1000 by 1000 cells)

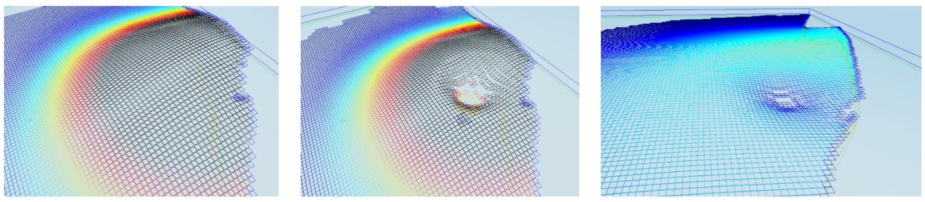


Figure 13: Removing oil over time: a negative $\Delta\eta$ is applyied on selected cells, effectively deleting the oil over time. This can be used to simulate de effect of a cleaning pump sucking the oil out of the water.

9.2 Encountered Issues

The main issue for the stability of the simulation is a build-up of oil at certain edges of the grid ("right" and "up" side of the grid). The problem is occurring during the first half time step, while computing u_x, u_y, v_x and v_y. During the calculation of these values, as we are on the edge of the grid, we use default values for the non-active cells used in the calculation (Section 7).

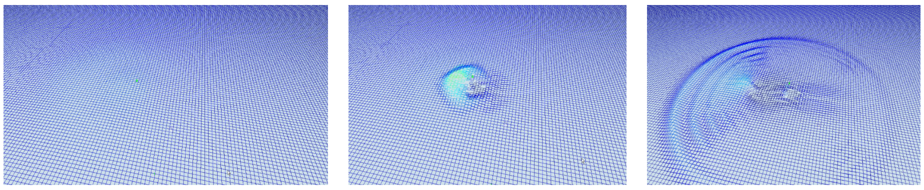


Figure 14: Push water over time: a velocity is enforced on selected cells, pushing the oil in a certain direction. This can be used to model the repeling force of a propeller.

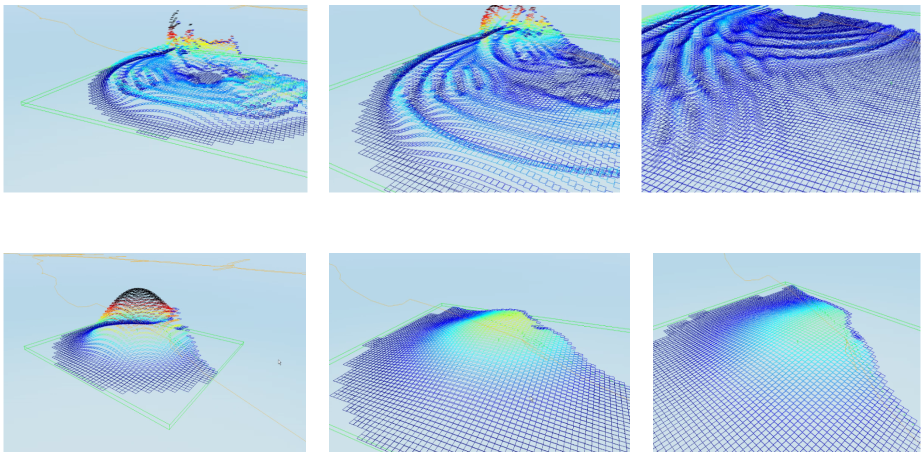


Figure 15: Effect of different viscosities: The top images represent oil with the same viscosity as water, while the bottom one the viscosity of thick tar. The spread of oil over the same timeframe becomes thus very different

The problems comes from the value returned by the default cell are not null (as empty cells should have 0 oil with null velocity). It results in a creation of oil that causes the oil slick to spread indefinitely in one direction.

A similar problem with the default cells also occurs during the calculation of the dynamic pressure using SEE. As SEE requires to calculate values 4 by 4, the dynamic pressure of 4 cells are calculated in parallel. In the case were the computation is made on a group of less than 4 cells, we fill in with default cells. The dynamic pressure function returned for such a case a NaN value for all cells, effectively bringing the simulation to an halt.

As the cells and grid system was made by VSTEP for their own water system, and reused for the bilayer SWE, the problem was raised but could not be solved. It also appeared that this build-up problem was present in the NAUTIS SWE water system, but not visible as their algorithm deleted active cells below a certain velocity/height threshold (deleting also a minimal negligible amount of water). This could however not be a solution for oil simulation, as we have a finite volume of oil that cannot be deleted.

Only a partial temporary fix was found, where disabling the usage of SSE would not cause the error, but it comes with a cost for performance and is thus not a desired fix

Figure 16 through 18 illustrate the problem, showing an oil slick propagating along one or both axis, having a build up of oil at its edges.

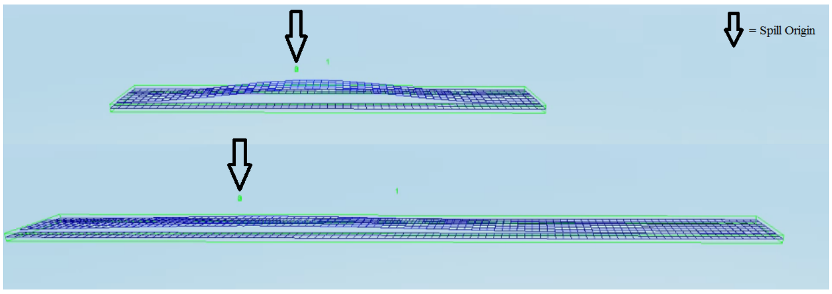


Figure 16: Oil slick propagating on 1 axis, showing the difference between the expected behaviour (left side) and the error induced by the build up (right side).

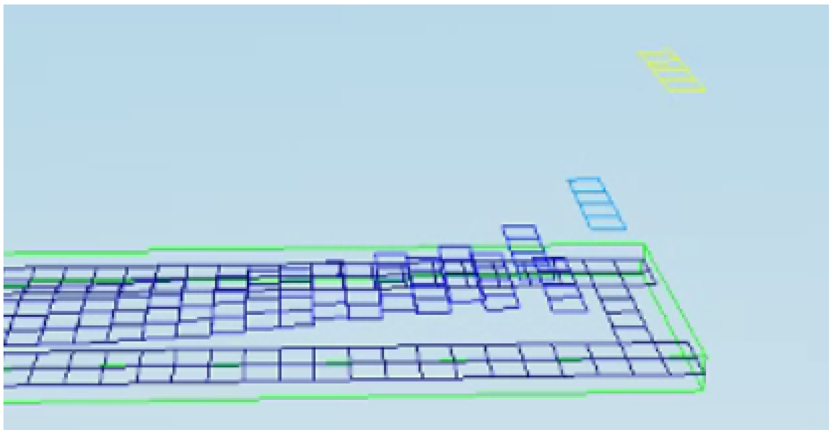


Figure 17: Zoom on the right side of the slick, showing oil ”creation”, the border cell creates oil and pass it to the inner cells.

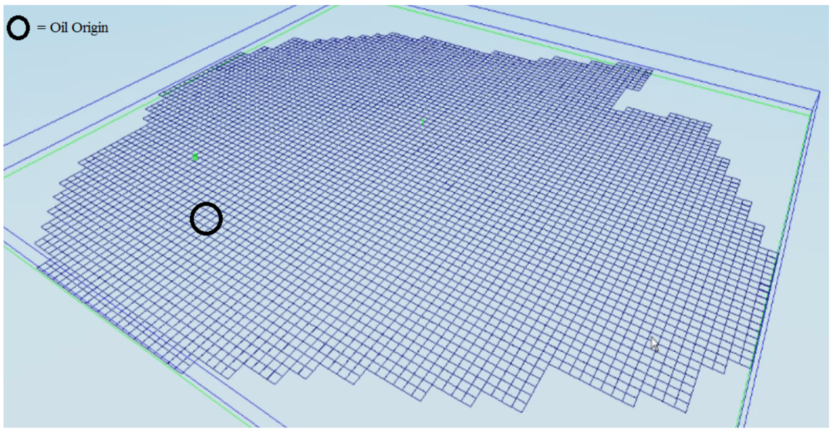


Figure 18: Oil slick propagating on both axis, showing its excessive spreading in one direction.

10 Conclusion and future work

10.1 Conclusion

The objective of this thesis was to simulate oil slicks on open ocean waters. We developed a modified version of the SWE to support 2 non-miscible fluids of different density, so that one (the oil slick) would always be atop the other. The robustness of the algorithm comes from the usage of a 2 step Lax-Wendroff solver proposed by [Sta99] for simulation of fluids under subsonic conditions (no "shocks" and hydraulic jumps), which fits our simulations conditions.

The algorithm was then implemented in the NAUTIS simulator as an interchangeable fluid system with their already existing SWE simulating their water environment. It was also developed to use SSE, saving computing time without altering the quality calculations, a critical element for real time simulation.

Other minor improvements where also made. For instance, improving object interaction with fluids, in order to avoid the Moses effect when having submerged objects.

The end result is a simulation of oil over water that can interact with its environment (boats, walls...) in a realistic manner, as well being added, pushed around and deleted in real time from the simulation (i.e during training, oil can be emitted from a leaking pipe, containned by booms and collected by pumps) as seen in Figure 13 to 15.

10.2 Limitation and future work

In its current state, the bi-layer SWE is very useful to model slicks on relatively calm waters (as cleaning training requires it). It has however still several limitations that could be addressed in future iterations of the algorithm:

- The question of visualisation is not touched here. Techniques used for SWE such as [HHL^{+}05] could be adapted for multiple fluids. At the moment, the default rendering are used here the cells outline changes height and color depending on a chosen variable (either water height, oil height, or both).
- The usage of automatic mesh refinement (AMR) to reduce simulation costs. This would be useful for larger operations, especially if we have several disparate oil slicks, where we could reduce the computational cost of less active (already well spread) and far from the user slicks. VSTEP is developing such an algorithm for their SWE, but it is not used as still under testing.
- Breaking waves are not possible with the classical SWE, [TMFSG07] proposes a solution for it. This would improve realism for places where beaching or crashing waves can appear and allow for wilder sea state and a wider range of trainings.
- Implementation of more effects, for example the droplets effects proposed by [OH95] adjusted for 2 fluids. This would permit better propagation of fluids when interacting with other objects (falling objects in the slick creating a splash, boat passing through, etc.)

- The grid and flag system could also be reviewed to avoid the computation error and oil build-up at certain edges. Maybe by starting with the simplier flag system of [MSJT08], deactivating computation on empty cells instead of deleting them.

References

[BH88] Ian Buchanan and Neil Hurford. Methods for predicting the physical changes in oil spilt at sea. *Oil and Chemical Pollution*, 4(4):311–328, 1988.

[BHN07] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics (TOG)*, 26(3):46, 2007.

[BHW13] Morten Bojsen-Hansen and Chris Wojtan. Liquid surface tracking with error compensation. *ACM Transactions on Graphics (TOG)*, 32(4):68, 2013.

[BMF07] Robert Bridson and Matthias Müller-Fischer. Fluid simulation: Siggraph 2007. In *ACM SIGGRAPH 2007 courses*, pages 1–81. ACM, 2007.

[CM10] Nuttapong Chentanez and Matthias Müller. Real-time simulation of large bodies of water with small scale details. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 197–206. Eurographics Association, 2010.

[CM11] Nuttapong Chentanez and Matthias Müller. Real-time eulerian water simulation using a restricted tall cell grid. *ACM Transactions on Graphics (TOG)*, 30(4):82, 2011.

[DL78] Robert A Dalrymple and Philip LF Liu. Waves over soft muds: a two-layer fluid model. *Journal of Physical Oceanography*, 8(6):1121–1131, 1978.

[DS88] Gerardus Athenasius Leonardus Delvigne and C_E Sweeney. Natural dispersion of oil. *Oil and Chemical Pollution*, 4(4):281–310, 1988.

[Fay71] James A Fay. Physical processes in the spread of oil on a water surface. In *International Oil Spill Conference*, volume 1971, pages 463–467. American Petroleum Institute, 1971.

[Fin99] Mervin F Fingas. The evaporation of oil spills: development and implementation of new prediction methodology. In *International Oil Spill Conference*, volume 1999, pages 281–287. American Petroleum Institute, 1999.

[FR86] Alain Fournier and William T Reeves. A simple model of ocean waves. *ACM Siggraph Computer Graphics*, 20(4):75–84, 1986.

[GDC08a] GDC. *Fast Water Simulation for Games Using Height Fields [Audio]*, `http://www.gdcvault.com/play/391/Fast-Water-Simulation-for-Games`, 02 2008. Accessed: 2016-07-26.

[GDC08b] GDC. *Fast Water Simulation for Games Using Height Fields [Slides]*, `http://www.gdcvault.com/play/203/Fast-Water-Simulation-for-Games`, 02 2008. Accessed: 2016-07-26.

[Gom00] Miguel Gomez. Interactive simulation of water surfaces. *Game programming gems*, 1:187–194, 2000.

[HHL$^+$05] Trond Runar Hagen, Jon M Hjelmervik, K-A Lie, Jostein R Natvig, and M Ofstad Henriksen. Visual simulation of shallow-water waves. *Simulation Modelling Practice and Theory*, 13(8):716–726, 2005.

[HNC02] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 161–166. ACM, 2002.

[HW65] Francis H Harlow and J Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

[KM90] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 49–57. ACM, 1990.

[KW06] Peter Kipfer and Rüdiger Westermann. Realistic and interactive simulation of rivers. In *Proceedings of Graphics Interface 2006*, pages 41–48. Canadian Information Processing Society, 2006.

[LeV02] Randall J LeVeque. Finite volume methods for hyperbolic problems. volume 31, chapter 12.9. Cambridge university press, 2002.

[LFBC84] WoJ Lehr, RJ Fraga, MS Belen, and HM Cekirge. A new technique to estimate initial spill size using a modified fay-type spreading formula. *Marine Pollution Bulletin*, 15(9):326–329, 1984.

[Lil61] DK Lilly. A proposed staggered-grid system for numerical integration of dynamic equations. *Monthly Feather Review*, 89(3):59–65, 1961.

[LZF10] Michael Lentine, Wen Zheng, and Ronald Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. In *ACM Transactions on Graphics (TOG)*, volume 29, page 114. ACM, 2010.

[MBMP80] D Mackay, I Buist, R Mascarenhas, and S Paterson. Oil spill processes and models. *Environment Canada Manuscript Report No. EE-8, Ottawa, Ontario*, 1980.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003.

[Mol11] Cleve B Moler. *Experiments with MATLAB*. Society for Industrial and Applied Mathematics, 2011.

[Mon92] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30:543–574, 1992.

[MSD07] Matthias Müller, Simon Schirm, and Stephan Duthaler. Screen space meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 9–15. Eurographics Association, 2007.

[MSJT08] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: class notes. In *ACM SIGGRAPH 2008 classes*, page 88. ACM, 2008.

[NKAS08] Muddassir Nazir, Faisal Khan, Paul Amyotte, and Rehan Sadiq. Multimedia fate of oil spills in a marine environment—an integrated modelling approach. *process safety and environmental protection*, 86(2):141–148, 2008.

[OH95] James F O'brien and Jessica K Hodgins. Dynamic simulation of splashing fluids. In *Computer Animation'95., Proceedings.*, pages 198–205. IEEE, 1995.

[PTB$^+$03] Simon Premžoe, Tolga Tasdizen, James Bigler, Aaron Lefohn, and Ross T Whitaker. Particle-based simulation of fluids. In *Computer Graphics Forum*, volume 22, pages 401–410. Wiley Online Library, 2003.

[Spa88] Malcolm L Spaulding. A state-of-the-art review of oil spill trajectory and fate modeling. *Oil and Chemical Pollution*, 4(1):39–55, 1988.

[SS95] P Sebastiao and C Guedes Soares. Modeling the fate of oil spills at sea. *Spill Science & Technology Bulletin*, 2(2):121–131, 1995.

[Sta99] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[Tes04] Jerry Tessendorf. Interactive water surfaces. *Game Programming Gems*, 4(265-274):8, 2004.

[TMFSG07] Nils Thurey, Matthias Muller-Fischer, Simon Schirm, and Markus Gross. Real-time breaking waves for shallow water simulations. In *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*, pages 39–46. IEEE, 2007.

[TWGT10] Nils Thürey, Chris Wojtan, Markus Gross, and Greg Turk. A multiscale approach to mesh-based surface tension flows. In *ACM Transactions on Graphics (TOG)*, volume 29, page 48. ACM, 2010.

[YHK07] Cem Yuksel, Donald H House, and John Keyser. Wave particles. In *ACM Transactions on Graphics (TOG)*, volume 26, page 99. ACM, 2007.

[ZB11] Jian Guo Zhou and Alistair GL Borthwick. Lattice boltzmann method for variable density shallow water equations. *Computers & Fluids*, 49(1):146–149, 2011.

[ZLO13] Jean De Dieu Zabsonré, Carine Lucas, and Adama Ouedraogo. Strong solutions for a 1d viscous bilayer shallow water model. *Nonlinear Analysis: Real World Applications*, 14(2):1216–1224, 2013.

[ZM13] Yubo Zhang and Kwan-Liu Ma. Spatio-temporal extrapolation for fluid animation. *ACM Transactions on Graphics (TOG)*, 32(6):183, 2013.

A Oil slick Estimation

Using [Fay71], this is the still used equation to estimate a spilled volume for the approximate shape of the slick:

$$A = \pi 0.98^2 \left\{ g V^2 t^{3/2} v_w^{-1/2} \left[\frac{\rho_w - \rho_o}{\rho_o} \right] \right\}^{1/3}$$

With A the area of the slick (m^2), V denotes initial volume of the slick (m^3), g denotes gravity constant (m/s^2), t denotes time (s), v_w denotes kinematic viscosity of water(m^2/s), ρ_w denotes water density (kg/m^3), ρ_o denotes oil density (kg/m^3). And a maximum area for A proposed in A to occur when the slick thickness is about $10^{-5}V^{3/4}$ [Fay71].

$$A = 10^5 V^{3/4}$$

But all this concerns a slick that is not subject to wind, this is corrected by [LFBC84] who divide the slick in an minor and major axis. He considers the slick as more or less an ellipse, having the major axis oriented in the direction of the wind. (A) thus become:

$$A = 2.27 \left[\frac{\rho_w - \rho_o}{\rho_o} \right]^{2/3} V^{2/3} t^{-1/2} + 0.04 \left[\frac{\rho_w - \rho_o}{\rho_o} \right]^{1/3} V^{1/3} W^{4/3} t$$

With W the wind (knots) and V the Volume in barrels (159 L/barrels).

B Oil Dispersion at Sea

The fraction of oil lost at sea can be expressed as follow:

$$D = D_a D_b$$

With D_a the fraction of sea surface lost at sea per hour, and D_b the fraction of dispersed oil not returning to the slick expressed as:

$$D_a = 0.11(W + 1)^2$$

$$D_b = (1 + 50\mu^{1/2} \delta s_t)^{-1}$$

With W the Wind speed ($m \cdot s^{-1}$), μ the viscosity (cP), δ the slick thickness (cm) and s_t the oil-water inter-facial tension ($dyne \cdot cm^{-1}$). [SS95] estimates the dispersion at in between $10\mu g.l^{-1}$ up to $2mg.l^{-1}$, justifying why it would be ignored in our simulation

C Calculation of the finite differences

This section presents how we came to the finite difference formulas for ΔV, $\Delta\rho$ and $\Delta\mu$, representing Volume, density and viscosity changes over one time step.

They are crucial as we used in the simulation obtained from the base equations

Volume change

Considering evaporation of the oil and incorporation of water inside the slick (the 2 behaviours tracked in this simulation), the volume occupied by the slick at time t is:

$$V_t = V_\theta - V_{Evaporated} + V_{WaterInOil}$$

We have from (7) the Fraction of oil in water Y, and from (5) & (6) the Volume evaporated E:

$$\begin{aligned} V_t &= V_\theta - V_\theta \times E_t + V_\theta * Y_t \\ &= V_\theta(1 - E_t + Y_t) \end{aligned}$$

Naturally t, $(t-1)$ and θ designate the current time step, the previous time step and the start conditions respectively.
We then obtain the ΔV by doing $V_t - V_{t-1}$:

$$\begin{aligned} \Delta V &= V_\theta(1 - E_t + Y_t) - [V_\theta(1 - E_{t-1} + Y_{t-1})] \\ &= V_\theta\,(Y_t - Y_{t-1})\,(E_{t-1} - E_t) \end{aligned}$$

At each time step we have thus calculate E_t & Y_t and reuse the data from the previous time step as E_{t-1} & Y_{t-1} to compute the changes.

Density change

Considering the same mechanism as for the volume change, the density at time t is:

$$\rho_t = \rho_{Evap}(1 - Y_t) + \rho_{water}(Y_t)$$

And we can replace ρ_{Evap} by:

$$\rho_{Evap} = \rho_{oil} + C_3(E_t)$$

Again, to use it in the simulation, we calculate $\Delta\rho$ by:

$$\begin{aligned} \Delta\rho &= \rho_t - \rho_{t-1} \\ &= \rho_{oil}(Y_{t-1} - Y_t) \quad +C_3(E_t(1 - Y_t) - E_{t-1}(1 - Y_{t-1}) \quad +\rho_{water}(Y_t - Y_{t-1}) \end{aligned}$$

At each time step we just reuse E and Y used in ΔV to compute the changes of density. We also remind that C_3 and C_4 are constant depending on the type of oil (see [3.1.3])

Viscosity change

Also considering only Evaporation and Emulsion:

$$\mu_t = \mu_{oil} \times exp\left[C_4 E_t\right] \times exp\left[\frac{2.5Y_t}{(1 - C_3 Y_t)}\right]$$

And again the derivative $\Delta\mu$

$$\begin{aligned}\Delta\mu &= \mu_{oil} \times exp\left[C_4 E_t\right] \times exp\left[\frac{2.5Y_t}{(1 - C_3 Y_t)}\right] - \mu_{oil} \times exp\left[C_4 E_{t-1}\right] \times exp\left[\frac{2.5Y_{t-1}}{(1 - C_3 Y_{t-1})}\right] \\ &= \mu_{oil}(exp\left[C_4 E_t + \frac{2.5Y_t}{(1 - C_3 Y_t)}\right] - exp\left[C_4 E_{t-1} + \frac{2.5Y_{t-1}}{(1 - C_3 Y_{t-1})}\right])\end{aligned}$$