Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

dusk & dawn - Shallow Water Exercise

DSL Training Workshop

Contents

- For this exercise, you will implement a simple **shallow water** using dusk & dawn
- You will combine the **FVM differential operators** from a previous exercise (gradient & divergence) as well as a simple **linear interpolation** technique to simulate a dynamic system simulation evolving surface waves

- Please note that the method we are going to implement is going to be of low order and dissipative! It is not at all intended to highlight the state of the art in solving the shallow water equation etc.

Shallow Water Equations

In climate simulation / numerical weather prediction the governing equations for the evolution of gravity waves are usually given as:

$$\frac{\partial \mathbf{v}}{\partial t} = -(f + \xi)\mathbf{k} \times \mathbf{v} - \nabla K + g\nabla h$$

$$\frac{\partial h}{\partial t} = -\nabla \cdot (\mathbf{v}h)$$

- **v**: velocity [m/s]
- h: height of fluid [m]
- k: local unit vector pointing upward [-
- f: Coriolis parameter [1/s]
- ξ: relative vorticity [1/s]
- K: kinetic energy per unit mass [J]

Shallow Water Equations

You stare at these equations for a while and realize that you don't want to deal with most of this stuff. Your friend that studies computer graphics assures you that these simplified equations are enough to simulate some waves

$$\frac{\partial \mathbf{v}}{\partial t} = g \nabla h$$

$$\frac{\partial h}{\partial t} = -h \nabla \cdot \mathbf{v}$$

- **v**: velocity [m/s]
- h: height of fluid [m]
- g: gravitational constant

Note: different sources assume different sign for gravitational acceleration. Here we assume g ≃ -9.8

Layton & van de Panne
https://doi.org/10.1007/s003710100131

SWE - Differential Operators

$$\frac{\partial \mathbf{v}}{\partial t} = g\boxed{\nabla h}$$

Gradient

$$\frac{\partial h}{\partial t} = -h\boxed{\nabla \cdot \mathbf{v}}$$

Divergence

MeteoSwiss

SWE - Differential Operators

$$\frac{\partial \mathbf{v}}{\partial t} = g \left\langle \nabla h \right\rangle_{\text{FVM}} \longrightarrow$$

```
grad_vx = sum_over(Cell > Edge,...
```

$$\frac{\partial h}{\partial t} = -h \left\langle \nabla \cdot \mathbf{v} \right\rangle_{\text{FVM}} \longrightarrow$$

```
div_v = sum_over(Cell > Edge,...
```

MeteoSwiss

SWE - Algorithm

Preprocess / Initialize

1. init height field on Cells
2. init velocity field on Cells

Time Stepper

1. save initial state
2. predictor

Spatial Derivatives

1. compute gradient

```
hC_x = sum_over(Edge>Cells, ...
```

2. compute divergence:

```
uvc_div = sum_over(Edge>Cells, ...
```

Boundary Conditions

1. Enforce reflective velocity boundaries
2. Enforce zero gradient at boundary

Build Up Equations

compute time derivatives

```
uC_t = Grav * hC_x...
```

Time Stepper

3. corrector

```
uC = uC_init + dt*uC_t
```

while:
$t < t_{final}$

SWE - Algorithm

will this work?

- all variables are co-located on the cells
- spatial derivatives propagate information from edges to cells
- edge values are never updated!

we need a means to update the edge values!

Time Stepper

1. save initial state
2. predictor

Spatial Derivatives

1. compute gradient
```
hC_x = sum_over(Cell>Edge, ...
```
2. compute divergence:
```
uvc_div = sum_over(Cell>Edge, ...
```

Boundary Conditions

1. Enforce reflective velocity boundaries
2. Enforce zero gradient at boundary

Build Up Equations

compute time derivatives
```
uC_t = Grav * hC_x...
```

Time Stepper

3. corrector
```
uC = uC_init + dt*uC_t
```

MeteoSwiss

SWE - Algorithm

will this work?

- all variables are co-located on the cells
- spatial derivatives propagate information from edges to cells
- edge values are never updated!

we need a means to update the edge values!

Time Stepper

1. save initial state
2. predictor

Interpolation

Linearly interpolate cell values to edges

Spatial Derivatives

1. compute gradient

```
hC_x = sum_over(Edge>Cells, ...
```

2. compute divergence:

```
uvc_div = sum_over(Edge>Cells, ...
```

Boundary Conditions

1. Enforce reflective velocity boundaries
2. Enforce zero gradient at boundary

Build Up Equations

compute time derivatives

```
uC_t = Grav * hC_x...
```

Time Stepper

3. corrector

```
uC = uC_init + dt*uC_t
```

MeteoSwiss

SWE - Interpolation

- You have seen that the diffusion solver from the last exercise was very dissipative
- Part of the reason for this was the very primitive interpolation from the edges to the vertices, which was a simple average
- For this exercise, let's try to do a little bit better and use linear interpolation in instead of simple averaging

- The necessary weights were pre computed for you in the variable alpha
- The `sum_over(Edge > Cell,...` primitive returns the cells in such a fashion that the first cell needs to be weighted by `1-alpha`, the second by `alpha`

SWE - Boundary Conditions

The boundary conditions are again quite simple:

- The velocity is zero at the boundary; waves are reflected

$$\mathbf{v}(e) = 0 \qquad |e \in \mathcal{B}e$$

- The surface gradient is zero at the boundaries

$$\langle \nabla h_c \rangle = 0 \qquad |c \in \mathcal{B}c$$

SWE - Final Algorithm

only implement this part this time around, time stepping is handled in the driver code!

Time Stepper

1. save initial state
2. predictor

Interpolation

Linearly interpolate cell values to edges

Spatial Derivatives

1. compute gradient

```
hC_x = sum_over(Edge>Cells, ...
```

2. compute divergence:

```
uvc_div = sum_over(Edge>Cells, ...
```

Boundary Conditions

1. Enforce reflective velocity boundaries
2. Enforce zero gradient at boundary

Build Up Equations

compute time derivatives

```
uC_t = Grav * hC_x...
```

Time Stepper

3. corrector

```
uC = uC_init + dt*uC_t
```

MeteoSwiss

Variable Reference I

An overview over all variables is given below. The ones in bold are out or "in-out" (both written to and being read from) variables. The others can be treated as read only.

`hC: Field[Cell]`, **`hC_t`**`: Field[Cell]`	height field on cells, temporal derivative of height field on cells
`vC: Field[Cell]`, **`vC_t`**`: Field[Cell]`	v component of velocity field on cells, temporal derivative of v component of velocity field on cells
`uC: Field[Cell]`, **`uC_t`**`: Field[Cell]`	u component of velocity field on cells, temporal derivative of u component of velocity field on cells
`hC_x``: Field[Cell]`, **`hC_y`**`: Field[Cell]`	x and y component of height field gradient on cells
`hE``: Field[Edge]`	height field on edges
`vE``: Field[Edge]`, **`uE`**`: Field[Edge]`	velocity field (u&v components) on edges
`nx: Field[Edge], ny: Field[Edge]`	cell normals on edges (x and y component)
`L: Field[Edge]`	edge lengths
`alpha: Field[Edge]`	linear interpolation coefficients to interp from edge to cell

MeteoSwiss

Variable Reference II

An overview over all variables is given below. The ones in bold are out or "in-out" (both written to and being read from) variables. The others can be treated as read only.

`boundary_edges: Field[Edge]`	`mask for boundary edges (true for boundary edges)`
`boundary_cells: Field[Cell]`	`mask for boundary cells (true for boundary cells)`
`A: Field[Cell]`	`area of cells`
`edge_orientation: Field[Cell > Edge]`	`sparse dimension that can be used to flip each normal locally outside. c.f. differential ops exercise`
`Grav: Field[Cell]`	`gravitational constant`

MeteoSwiss

Hints

- Hints are on the next slides
- Please consider them only if you're seriously stuck
- Some general hints are on the next slides, on the slide after that parts of the solution are revealed

Hints

- dusk & dawn has no 2d (or 3d, for that matter) vector type (yet). u,v denote the two components of the vector **v**
- The gradient and divergence consume the geometric quantities given on the edges
  - `edge_orientation` and `L` are used by both the gradient and the divergence
  - the only difference between the _x and _y part of the gradient is the component of the normal they consume
- take care to use the correct signs when building up the ODEs

Hints - Skeleton

```
with levels_downward:
    # lerp cell quantities to edges
    hE =
    uE =
    vE =
    # boundary conditions on cells
    if (boundary_edges):
        uE = 0.
        vE = 0.
    # height field gradient
    hC_x =
    hC_y =
    # height field gradient is zero on the boundaries
    if (boundary_cells):
        hC_x = 0.
        hC_y = 0.
    # divergence of velocity field
    uvC_div =
    # build ODE's
    uC_t =
    vC_t =
    hC_t =
```