Accepted Manuscript
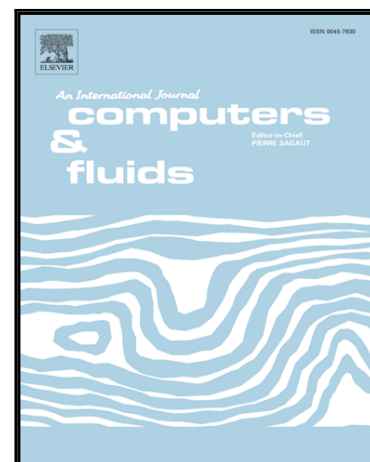
GPU Driven Finite Difference WENO Scheme for Real Time Solution of the Shallow Water Equations

P. Parna, K. Meyer, R. Falconer

Please cite this article as: P. Parna, K. Meyer, R. Falconer, GPU Driven Finite Difference WENO Scheme for Real Time Solution of the Shallow Water Equations, *Computers and Fluids* (2017), doi: 10.1016/j.compfluid.2017.11.012

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Highlights

- A new 3rd order in space and 2nd order in time finite difference scheme is proposed.

- Work is based on Picard integral formulation coupled with WENO reconstruction.

- Strategies are presented that allow for a fast single kernel GPU implementation.

- Results indicate single precision arithmetic to be sufficient for the formulation.

- Simultaneous simulation and rendering is achieved on consumer-level hardware.

1

GPU Driven Finite Difference WENO Scheme for Real Time Solution of the Shallow Water Equations

P. Parna[a,*], K. Meyer[a], R. Falconer[a]

[a]*Abertay University, Bell Street, Dundee DD1 1HG, Scotland*

Abstract

The shallow water equations are applicable to many common engineering problems involving modelling of waves dominated by motions in the horizontal directions (e.g. tsunami propagation, dam breaks). As such events pose substantial economic costs, as well as potential loss of life, accurate real-time simulation and visualization methods are of great importance. For this purpose, we propose a new finite difference scheme for the 2D shallow water equations that is specifically formulated to take advantage of modern GPUs. The new scheme is based on the so-called Picard integral formulation of conservation laws combined with Weighted Essentially Non-Oscillatory reconstruction. The emphasis of the work is on third order in space and second order in time solutions (in both single and double precision). Further, the scheme is well-balanced for bathymetry functions that are not surface piercing and can handle wetting and drying in a GPU-friendly manner without resorting to long and specific case-by-case procedures. We also present a fast single kernel GPU implementation with a novel boundary condition application technique that allows for simultaneous real-time visualization and single precision simulations even on large ($> 2000 \times 2000$) grids on consumer-level hardware - the full kernel source codes are also provided online at https://github.com/pparna/swe_pifweno3.

Keywords: shallow water, Picard integral, WENO, finite difference, GPGPU

*Corresponding author
Email address: p.parna@phys-gfx.net (P. Parna)
URL: www.phys-gfx.net (P. Parna)

1. Introduction

The shallow water equations (SWE) are a set of hyperbolic partial differential equations that arise from the more general inviscid Navier-Stokes equations (also referred to as the Euler equations) under the assumption that the vertical water depth h_0 is much smaller than the horizontal length scale L of the waves, i.e. $h_0 \ll L$, and hence the vertical acceleration is considered negligible [1, p.89,91]. Such a simplification is especially beneficial from a computation point of view as the arising equations result in dimensional reduction from $\mathbb{R}^3$ to $\mathbb{R}^2$, while still describing the evolution of a three dimensional fluid surface. As a result, the equations are often used for real-time flood prediction [2], simulations of tsunami propagation and inundation [3], modelling of dam breaks [4] and even computer graphics animations of water [5].

The shallow water equations in 2D conservation form are given as:

$$\frac{\partial \boldsymbol{U}(x,y,t)}{\partial t} + \frac{\partial \boldsymbol{F}(\boldsymbol{U}(x,y,t))}{\partial x} + \frac{\partial \boldsymbol{G}(\boldsymbol{U}(x,y,t))}{\partial y} = \boldsymbol{S}(b(x,y)) \qquad (1)$$

where $\boldsymbol{U}$ is the vector of conserved variables (mass and momentum), $\boldsymbol{F}$ and $\boldsymbol{G}$ the x and y directional fluxes, respectively; $\boldsymbol{S}$ is the source term due to topography underneath the water surface (also referred to as the bathymetry). In this work, we are interested in modelling the time-independent source term (i.e. only static bathymetry functions are considered). The vectors themselves are given as:

$$\boldsymbol{U} = \begin{pmatrix} h \\ hu \\ hv \end{pmatrix} ; \ \boldsymbol{F} = \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{pmatrix} ; \ \boldsymbol{G} = \begin{pmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{pmatrix} ; \ \boldsymbol{S} = \begin{pmatrix} 0 \\ -gh\dfrac{\partial b}{\partial x} \\ -gh\dfrac{\partial b}{\partial y} \end{pmatrix}$$
$$(2)$$

where h is the water height, u and v are the horizontal velocities, b is the underlying topography function and g the gravitational constant. It will also be useful to consider the total surface elevation $\eta = b + h$ as illustrated in Figure 1.
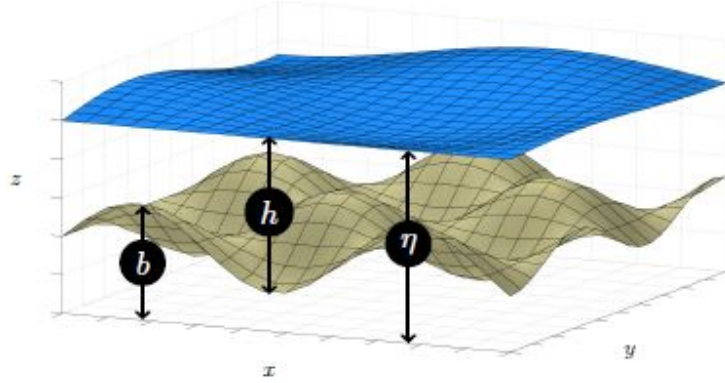
3

Figure 1: Spatial setup for the shallow water equations.

The conservation law form of the shallow water equations lends itself to many well-known numerical methods - the equations have been solved by several authors using classic finite difference schemes, such as the MacCormack method [3, 6, 7], alongside specifically designed finite volume schemes such as the central-upwind scheme by Kurganov and Petrova [8, 4, 9]. These schemes generally have a fixed order of accuracy - often the spatial order of accuracy is formulated to second order, with no straight-forward way of increasing it. Furthermore, these schemes commonly use the method of lines (MOL) approach[1] with Runge-Kutta integration for timestepping, leading to multi-step implementations requiring complex flux evaluations at every stage. One of the most common arbitrary-order finite difference methods is based on the Weighted Essentially Non-Oscillatory (WENO) reconstruction procedures [10]. Recently, the application of WENO to conservation laws was further modified to incorporate time-averaged flux functions by Seal *et al.* [11] (called the Picard Integral Formulation of WENO (PIFWENO) schemes), who also successfully applied the idea to the compressible Euler equations [12]. The compact nature of the

---

[1]A partial differential equation is transformed into multiple ordinary differential equations via initial semi-discretization in space - this results in n ordinary differential equations in time where n is the total number of grid cells.

4

PIFWENO formulation (specifically the Taylor timestepping variation) makes it a particularly interesting candidate for high-accuracy real-time simulations and as such the following paper provides a complete derivation and description of a PIFWENO-type scheme for the 2D shallow water equations that is third

25 order accurate in space and second order accurate in time, well-balanced (the flux terms balance the source term [9]) and is capable of retaining the positivity of the water depth, thus allowing for simulations with dry zones. Further, we present an optimized, single pass GPU implementation of the scheme capable of achieving real-time performance on various grid sizes using either single or

30 double precision floating point arithmetic.

The rest of the paper is organized as follows: in Section 2 a detailed mathematical description of the numerical scheme is presented, followed by an overview of the practical implementation describing the employed optimization strategies in Section 3. In Section 4 the numerical accuracy and the capability of the scheme

35 to model complex flows with moving shorelines are investigated, alongside verification of the well-balanced property and grid convergence rates. Furthermore, the performance of the GPU implementation for real-time computation and rendering is assessed using both single and double precision arithmetic. Finally, increasing the scheme's spatial and temporal orders are discussed, followed by

40 conclusions of the undertaken research in Section 5.

2. Numerical Method

2.1. Picard Integral Formulation for SWE

The Picard integral formulation (PIF) of the SWE can be defined by integrating Equation (1) over the interval $t \in [t^n, t^{n+1}]$ [11] (subscripts denote derivatives while superscripts denote the **time level**):

$$\boldsymbol{U}^{n+1} = \boldsymbol{U}^n - \Delta t \widetilde{\boldsymbol{F}}_x^n - \Delta t \widetilde{\boldsymbol{G}}_y^n + \Delta t \boldsymbol{S}^n \tag{3}$$

5

where $\widetilde{\boldsymbol{F}}^n$ and $\widetilde{\boldsymbol{G}}^n$ are the time-averaged fluxes defined as:

$$\widetilde{\boldsymbol{F}}^n = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} \boldsymbol{F}^n(\boldsymbol{U})\, dt$$

$$\widetilde{\boldsymbol{G}}^n = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} \boldsymbol{G}^n(\boldsymbol{U})\, dt. \tag{4}$$

The conservative finite difference discretization of Equation (3) can be written as:

$$\boldsymbol{U}_{i,j}^{n+1} = \boldsymbol{U}_{i,j}^n - \frac{\Delta t}{\Delta x}\left(\widehat{\boldsymbol{F}}_{i+1/2,j}^n - \widehat{\boldsymbol{F}}_{i-1/2,j}^n\right) - \frac{\Delta t}{\Delta y}\left(\widehat{\boldsymbol{G}}_{i,j+1/2}^n - \widehat{\boldsymbol{G}}_{i,j-1/2}^n\right) + \Delta t \boldsymbol{S}_{i,j}^n \tag{5}$$

where the values of $\widehat{\boldsymbol{F}}^n$ and $\widehat{\boldsymbol{G}}^n$ at the cell edges are given by the WENO reconstruction procedure of the time-averaged fluxes $\widetilde{\boldsymbol{F}}^n$ and $\widetilde{\boldsymbol{G}}^n$, respectively. The time averaged fluxes can be approximated via Taylor expansion of the fluxes centred at $t = t^n$ and then integrating the result with respect to t [11] (henceforward, **time level** n dropped for convenience):

$$\widetilde{\boldsymbol{F}} = \boldsymbol{F}(\boldsymbol{U}) + \frac{\Delta t}{2}\frac{d\boldsymbol{F}(\boldsymbol{U})}{dt} + \mathcal{O}(\Delta t^2)$$

$$\widetilde{\boldsymbol{G}} = \boldsymbol{G}(\boldsymbol{U}) + \frac{\Delta t}{2}\frac{d\boldsymbol{G}(\boldsymbol{U})}{dt} + \mathcal{O}(\Delta t^2). \tag{6}$$

Higher order approximations can be achieved by including more terms in the Taylor expansions. However, these require evaluations of Hessians and other higher order tensors which grow exponentially in size [11] - we found second order to be sufficient for our purposes. Note that the Hessian tensors of the flux functions for the SWE involve a scalar multiplier $1/h$ which further complicates simulations involving dry zones ($h = 0$). The temporal derivatives appearing in Equation (6) can be expanded as:

$$\frac{d\boldsymbol{F}(\boldsymbol{U})}{dt} = \frac{\partial \boldsymbol{F}(\boldsymbol{U})}{\partial \boldsymbol{U}}\frac{\partial \boldsymbol{U}}{\partial t}$$

$$\frac{d\boldsymbol{G}(\boldsymbol{U})}{dt} = \frac{\partial \boldsymbol{G}(\boldsymbol{U})}{\partial \boldsymbol{U}}\frac{\partial \boldsymbol{U}}{\partial t} \tag{7}$$

6

where $\partial\boldsymbol{F}/\partial\boldsymbol{U}$ and $\partial\boldsymbol{G}/\partial\boldsymbol{U}$ are the flux Jacobians. For the SWE these are [13, p.429]:

$$\frac{\partial\mathbf{F}}{\partial\mathbf{U}} = \begin{pmatrix} 0 & 1 & 0 \\ -u^2 + gh & 2u & 0 \\ -uv & v & u \end{pmatrix} \qquad \frac{\partial\mathbf{G}}{\partial\mathbf{U}} = \begin{pmatrix} 0 & 0 & 1 \\ -uv & v & u \\ -v^2 + gh & 0 & 2v \end{pmatrix}. \qquad (8)$$

Combining Equation (7) with the Cauchy-Kowalewski procedure (using the original PDE in Equation (1)) and plugging the results into Equation (6) gives the final form of the time-averaged fluxes as:

$$\begin{aligned} \widetilde{\boldsymbol{F}}^n &= \boldsymbol{F}(\boldsymbol{U}) + \frac{\Delta t}{2}\frac{\partial\boldsymbol{F}(\boldsymbol{U})}{\partial\boldsymbol{U}}\left(\boldsymbol{S} - \boldsymbol{F}(\boldsymbol{U})_x - \boldsymbol{G}(\boldsymbol{U})_y\right) \\ \widetilde{\boldsymbol{G}}^n &= \boldsymbol{G}(\boldsymbol{U}) + \frac{\Delta t}{2}\frac{\partial\boldsymbol{G}(\boldsymbol{U})}{\partial\boldsymbol{U}}\left(\boldsymbol{S} - \boldsymbol{F}(\boldsymbol{U})_x - \boldsymbol{G}(\boldsymbol{U})_y\right). \end{aligned} \qquad (9)$$

Any derivatives that appear in Equation (9) are evaluated using simple central finite difference equations of order $k - 1$ where k is the order of the WENO reconstruction. Due to the extra Δt term, this approximation is sufficiently accurate (for any higher order time-averaged flux approximations, every higher order term can be evaluated with a consequently lower order approximation stencil, e.g. see [11] for an example of third order approximation).

2.2. WENO Reconstruction

The core idea of the essentially non-oscillatory (ENO) reconstruction procedure [14] is to choose an approximation to the function to be reconstructed such that it is as smooth as possible in the candidate stencil used for the approximation. Weighted ENO (WENO) [10] takes this a step further by instead of choosing a single approximation, combining all of the possible rth order approximations with appropriate weightings whereby smoother approximations in a given stencil receive larger weights. This results in schemes that are of $(2r - 1)$th order accurate in smooth regions. For the sake of completeness, we briefly summarize the procedure here - a more in-depth introduction can be found in the report by Shu [15].

The first step for finite difference WENO methods is to split the fluxes into

7

positive and negative parts using a smooth flux splitting in order to follow correct upwinding [16] (henceforth only $\widetilde{\boldsymbol{F}}$ is considered but the same applies to $\widetilde{\boldsymbol{G}}$):

$$\widehat{\boldsymbol{F}}_{i+1/2,j} = \boldsymbol{f}^+(\boldsymbol{U}) + \boldsymbol{f}^-(\boldsymbol{U}) \tag{10}$$

where

$$\frac{\partial \boldsymbol{f}^+(\boldsymbol{U})}{\partial \boldsymbol{U}} \geq 0; \quad \frac{\partial \boldsymbol{f}^-(\boldsymbol{U})}{\partial \boldsymbol{U}} \leq 0. \tag{11}$$

One of the most common flux splittings satisfying Equation (11) is the Lax-Friedrichs flux splitting [16]:

$$\boldsymbol{f}^\pm(\boldsymbol{U}) = \frac{1}{2}\left(\widetilde{\boldsymbol{F}}(\boldsymbol{U}) \pm \alpha \boldsymbol{U}\right) \tag{12}$$

where $\alpha = \max\limits_{\boldsymbol{U} \in \mathcal{I}} \max\limits_{1 \leq i \leq 3} |\lambda_i|$ with λ_i being the ith eigenvalue of the flux Jacobian and $\mathcal{I}$ the stencil of values to be considered. Global splitting is achieved when $\mathcal{I}$ includes the entire computational domain whereas defining a local stencil **results in the Rusanov flux formulation [17] (also referred to as the local Lax-Friedrichs (LLF) flux splitting [13, p.233]).** In this work, LLF is used and as such, e.g. in the x direction the third order stencil is given as $\mathcal{I} \in \{\boldsymbol{U}_{i-1,j}, \boldsymbol{U}_{i,j}, \boldsymbol{U}_{i+1,j}, \boldsymbol{U}_{i+2,j}\}$.

Generally, a more robust scheme is achieved by projecting the fluxes and conserved variables to the local characteristic fields before applying the WENO reconstruction procedures [18]. For this, the left and right eigenvector-matrices need to be defined at the flux interfaces: a simple average of the two states beside the interface is sufficient for this. For example, in the x direction:

$$\begin{aligned}
\boldsymbol{U}_{i+1/2,j} &= \frac{1}{2}\left(\boldsymbol{U}_{i,j} + \boldsymbol{U}_{i+1,j}\right) \\
R^{-1}_{i+1/2,j} &= R^{-1}(\boldsymbol{U}_{i+1/2,j}) \\
R_{i+1/2,j} &= R(\boldsymbol{U}_{i+1/2,j}).
\end{aligned} \tag{13}$$

8

Defining $c = \sqrt{gh}$, the left and right eigenvector-matrices for the SWE are given as [19]:

$$
R_F^{-1} = \begin{pmatrix} \dfrac{u+c}{2c} & -\dfrac{1}{2c} & 0 \\ -v & 0 & 1 \\ -\dfrac{u-c}{2c} & \dfrac{1}{2c} & 0 \end{pmatrix} \qquad R_F = \begin{pmatrix} 1 & 0 & 1 \\ u-c & 0 & u+c \\ v & 1 & v \end{pmatrix}
$$

$$
R_G^{-1} = \begin{pmatrix} \dfrac{v+c}{2c} & 0 & -\dfrac{1}{2c} \\ -u & 1 & 0 \\ -\dfrac{v-c}{2c} & 0 & \dfrac{1}{2c} \end{pmatrix} \qquad R_G = \begin{pmatrix} 1 & 0 & 1 \\ u & 1 & u \\ v-c & 0 & v+c \end{pmatrix}.
$$

(14)

Therefore, Equation (12) becomes:

$$
\boldsymbol{f}^{\pm}(\boldsymbol{U}) = \frac{1}{2}\left(R_F^{-1}\widetilde{\boldsymbol{F}}(\boldsymbol{U}) \pm \alpha_i R_F^{-1}\boldsymbol{U} \right) \tag{15}
$$

where the dissipation coefficient α can now be chosen based on the maximum values in the specific characteristic field [17], i.e. $\alpha_i = \max\limits_{\boldsymbol{U}\in\mathcal{I}} |\lambda_i|$ where i is the ith characteristic variable. After the reconstruction, the values can be projected back to physical space using R:

$$
\widehat{\boldsymbol{F}}_{i+1/2,j} = R_F \cdot (\boldsymbol{f}^{+}(\boldsymbol{U}) + \boldsymbol{f}^{-}(\boldsymbol{U})). \tag{16}
$$

50   Finally, the WENO reconstruction can be applied on the split fluxes: $\boldsymbol{f}^{+}(\boldsymbol{U})$ uses a stencil biased to the left and $\boldsymbol{f}^{-}(\boldsymbol{U})$ uses one biased to the right [16]. The third order reconstruction procedure is given in Appendix A.

2.3. Well-Balanced Treatment of the Source Term

The well-balanced treatment of the source term, also referred to as the exact conservation property (C-property), is achieved with the incorporation of the method by Xing and Shu [20] into the formulation. The core of the idea is to

9

first split the source term into two parts as follows:

$$\boldsymbol{S} = \begin{pmatrix} 0 \\ \left(\dfrac{1}{2}gb^2\right)_x \\ \left(\dfrac{1}{2}gb^2\right)_y \end{pmatrix} + \begin{pmatrix} 0 \\ -g(h+b)b_x \\ -g(h+b)b_y \end{pmatrix}. \tag{17}$$

Next, since b is independent of the time t, then the Lax-Friedrichs flux can be modified to include η:

$$\boldsymbol{f}^{\pm}(\boldsymbol{U}) = \frac{1}{2}\left(\widetilde{\boldsymbol{F}}(\boldsymbol{U}) \pm \alpha \begin{pmatrix} \eta \\ hu \\ hv \end{pmatrix}\right). \tag{18}$$

This replacement is useful because for still water stationary solutions, η remains constant. Due to the flux splitting, the source term derivatives are further split into two parts, each of which can be associated with the reconstruction of either $\boldsymbol{f}^{-}(\boldsymbol{U})$ or $\boldsymbol{f}^{+}(\boldsymbol{U})$:

$$\begin{aligned} \begin{pmatrix} 0 \\ \left(\frac{1}{2}gb^2\right)_x \\ \left(\frac{1}{2}gb^2\right)_y \end{pmatrix} &= \frac{1}{2}\begin{pmatrix} 0 \\ \left(\frac{1}{2}gb^2\right)_x \\ \left(\frac{1}{2}gb^2\right)_y \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 \\ \left(\frac{1}{2}gb^2\right)_x \\ \left(\frac{1}{2}gb^2\right)_y \end{pmatrix} \\ \begin{pmatrix} 0 \\ b_x \\ b_y \end{pmatrix} &= \frac{1}{2}\begin{pmatrix} 0 \\ b_x \\ b_y \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 \\ b_x \\ b_y \end{pmatrix}. \end{aligned} \tag{19}$$

55 The further split source components are then approximated using the WENO procedure with the non-linear weights fixed with respect to the flux being reconstructed. This means that the usual small stencils are used to approximate the source term derivatives, followed by a combination of said stencils using the same weights computed for the relevant flux reconstructions. Xing and Shu [20] 60 further suggest absorbing the computation of the first source term into the computation of the relevant flux (note that smoothness indicators would still be based on the flux only) - this was found to be suitable for the component-by-component WENO solution, however the characteristic projection based scheme

10

was found to exhibit worse stability in simulations involving dry regions. Hence, for the characteristic version we chose to reconstruct both of the source terms separately. Further, in order to achieve the well-balanced property, the central finite difference approximations of the source term in Equation (9) must also follow the splitting given in Equation (17).

2.4. General Handling of the Wetting/Drying Processes

For the scheme to allow for wetting and drying processes, the computation of the mass-conservation equation must be made positivity preserving - physically, the values of h need to remain non-negative; numerically the eigenvalues of **the flux functions must be real in order to retain hyperbolicity of the system** [13, p.425-426]. For this purpose, the maximum principle preserving method of Liang and Xu [21] can be incorporated for the mass computation equations. To this end, the final fluxes for the mass can be re-written as:

$$\widetilde{f}_{i+1/2,j} = \theta_{i+1/2,j}\left(\widehat{F}^{(1)}_{i+1/2,j} - \hat{f}_{i+1/2,j}\right) + \hat{f}_{i+1/2,j}$$
$$\widetilde{g}_{i,j+1/2} = \theta_{i,j+1/2}\left(\widehat{G}^{(1)}_{i,j+1/2} - \hat{g}_{i,j+1/2}\right) + \hat{g}_{i,j+1/2} \tag{20}$$

where $\widehat{F}^{(1)}_{i+1/2,j}$ and $\widehat{G}^{(1)}_{i,j+1/2}$ are the first entries of $\widehat{\boldsymbol{F}}_{i+1/2,j}$ and $\widehat{\boldsymbol{G}}_{i+1/2,j}$, respectively. The $\hat{f}_{i+1/2,j}$ and $\hat{g}_{i,j+1/2}$ are first order monotone flux approximations that preserve the strict maximum principle. These can be evaluated using the Lax-Friedrichs flux [12]:

$$\hat{f}_{i+1/2,j} = \frac{1}{2}\left(\boldsymbol{F}(\boldsymbol{U})_{i,j} + \boldsymbol{F}(\boldsymbol{U})_{i+1,j} - \alpha_i\left(\boldsymbol{U}_{i+1,j} - \boldsymbol{U}_{i,j}\right)\right)$$
$$\hat{g}_{i,j+1/2} = \frac{1}{2}\left(\boldsymbol{G}(\boldsymbol{U})_{i,j} + \boldsymbol{F}(\boldsymbol{U})_{i,j+1} - \alpha_j\left(\boldsymbol{U}_{i,j+1} - \boldsymbol{U}_{i,j}\right)\right) \tag{21}$$

where $\alpha_k = \max\limits_{\boldsymbol{U}\in\mathcal{I}}\max\limits_{1\leq i\leq 3}|\lambda_i|$. The $\theta_{i+1/2,j}$ and $\theta_{i,j+1/2}$ in Equation (20) are the parameters to be found such that the interpolation between the high order WENO flux and the low-order LF flux is of as high order as possible while preserving the positivity of the solution. Thus, we need to find values

$$0 \leq \Lambda_{L,i,j}; \Lambda_{R,i,j}; \Lambda_{D,i,j}; \Lambda_{U,i,j} \leq 1 \tag{22}$$

11

such that:

$$\left(\theta_{i-1/2,j}, \theta_{i+1/2,j}, \theta_{i,j-1/2}, \theta_{i,j+1/2}\right) \in [0, \Lambda_{L,i,j}] \times [0, \Lambda_{R,i,j}] \times [0, \Lambda_{D,i,j}] \times [0, \Lambda_{U,i,j}].$$
(23)

Plugging Equation (20) into Equation (5), we have for the h component (let $\lambda_x = \Delta t / \Delta x$ and $\lambda_y = \Delta t / \Delta y$):

$$\begin{aligned}
h_{i,j}^{n+1} = h_{i,j}^n &- \lambda_x \left(\hat{f}_{i+1/2,j} - \hat{f}_{i-1/2,j}\right) - \lambda_y \left(\hat{g}_{i,j+1/2} - \hat{g}_{i,j-1/2}\right) \\
&- \lambda_x \theta_{i+1/2,j} \left(\widehat{F}_{i+1/2,j}^{(1)} - \hat{f}_{i+1/2,j}\right) + \lambda_x \theta_{i-1/2,j} \left(\widehat{F}_{i-1/2,j}^{(1)} - \hat{f}_{i-1/2,j}\right) \\
&- \lambda_y \theta_{i,j+1/2} \left(\widehat{G}_{i,j+1/2}^{(1)} - \hat{g}_{i,j+1/2}\right) + \lambda_y \theta_{i,j-1/2} \left(\widehat{G}_{i,j-1/2}^{(1)} - \hat{g}_{i,j-1/2}\right).
\end{aligned}$$
(24)

Defining:

$$\Gamma_{i,j} = -\left(h_{i,j}^n - \lambda_x \left(\hat{f}_{i+1/2,j} - \hat{f}_{i-1/2,j}\right) - \lambda_y \left(\hat{g}_{i,j+1/2} - \hat{g}_{i,j-1/2}\right)\right)$$
(25)

and

$$\begin{aligned}
\mathcal{F}_{i+1/2,j} &= -\lambda_x \left(\widehat{F}_{i+1/2,j}^{(1)} - \hat{f}_{i+1/2,j}\right) \\
\mathcal{F}_{i-1/2,j} &= \lambda_x \left(\widehat{F}_{i-1/2,j}^{(1)} - \hat{f}_{i-1/2,j}\right) \\
\mathcal{F}_{i,j+1/2} &= -\lambda_y \left(\widehat{G}_{i,j+1/2}^{(1)} - \hat{g}_{i,j+1/2}\right) \\
\mathcal{F}_{i,j-1/2} &= \lambda_y \left(\widehat{G}_{i,j-1/2}^{(1)} - \hat{g}_{i,j-1/2}\right)
\end{aligned}$$
(26)

gives us:

$$h_{i,j}^{n+1} = -\Gamma_{i,j} + \theta_{i+1/2,j}\mathcal{F}_{i+1/2,j} + \theta_{i-1/2,j}\mathcal{F}_{i-1/2,j} + \theta_{i,j+1/2}\mathcal{F}_{i,j+1/2} + \theta_{i,j-1/2}\mathcal{F}_{i,j-1/2}.$$
(27)

We require $h_{i,j}^{n+1} \geq 0$, hence we need to solve:

$$\theta_{i+1/2,j}\mathcal{F}_{i+1/2,j} + \theta_{i-1/2,j}\mathcal{F}_{i-1/2,j} + \theta_{i,j+1/2}\mathcal{F}_{i,j+1/2} + \theta_{i,j-1/2}\mathcal{F}_{i,j-1/2} - \Gamma_{i,j} \geq 0.$$
(28)

This can be solved by checking the signs of the values in Equation (26) and collectively defining $\Lambda_{L,i,j}, \Lambda_{R,i,j}, \Lambda_{D,i,j}$ and $\Lambda_{U,i,j}$. For example if $\mathcal{F}_{i+1/2,j} \geq 0$, $\mathcal{F}_{i-1/2,j} < 0$, $\mathcal{F}_{i,j+1/2} < 0$ and $\mathcal{F}_{i,j-1/2} \geq 0$ then:

$$\begin{cases}
\Lambda_{R,i,j} = \Lambda_{D,i,j} = 1 \\
\Lambda_{L,i,j} = \Lambda_{U,i,j} = \min\left(1, \dfrac{\Gamma_{i,j}}{\mathcal{F}_{i-1/2,j} + \mathcal{F}_{i,j+1/2}}\right).
\end{cases}$$
(29)

There are a total of 16 cases to consider. A closer look at the solutions to Equation (28) reveals a clear pattern and hence we propose an alternative, **minimal case** formulation instead:

$$\Lambda_{R,i,j} = (1 - \alpha) + \alpha Q$$
$$\Lambda_{L,i,j} = (1 - \beta) + \beta Q$$
$$\Lambda_{U,i,j} = (1 - \gamma) + \gamma Q \tag{30}$$
$$\Lambda_{D,i,j} = (1 - \delta) + \delta Q$$

where

$$Q = \min\left(1, \frac{\Gamma_{i,j}}{\alpha \mathcal{F}_{i+1/2,j} + \beta \mathcal{F}_{i-1/2,j} + \gamma \mathcal{F}_{i,j+1/2} + \delta \mathcal{F}_{i,j-1/2}}\right) \tag{31}$$

and

$$\alpha = \begin{cases} 1 & \text{if } \mathcal{F}_{i+1/2,j} < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\beta = \begin{cases} 1 & \text{if } \mathcal{F}_{i-1/2,j} < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma = \begin{cases} 1 & \text{if } \mathcal{F}_{i,j+1/2} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

$$\delta = \begin{cases} 1 & \text{if } \mathcal{F}_{i,j-1/2} < 0 \\ 0 & \text{otherwise.} \end{cases}$$

Note that in actual implementation branching can be completely avoided by directly utilizing the return value of a conditional statement. Further, if $\alpha = \beta = \gamma = \delta = 0$ then one can take $Q = 0$.

Finally, the local limiting parameters are given as:

$$\theta_{i+1/2,j} = \min\left(\Lambda_{R,i,j}, \Lambda_{L,i+1,j}\right)$$
$$\theta_{i,j+1/2} = \min\left(\Lambda_{U,i,j}, \Lambda_{D,i,j+1}\right). \tag{33}$$

Besides positivity preservation, the velocities at the end of a simulation timestep need to be desingularized in order to avoid high velocities developing near

13

the wet/dry interface that inevitably would lead to instabilities. This can be achieved using the method by [8]: the following equations are applied in regions where $h_{i,j}$ is less than a specified threshold value ϵ:

$$
\begin{aligned}
u_{i,j} &= \frac{\sqrt{2}h_{i,j}(hu)_{i,j}}{\sqrt{h_{i,j}^4 + \max(h_{i,j}^4, \epsilon)}} \\
v_{i,j} &= \frac{\sqrt{2}h_{i,j}(hv)_{i,j}}{\sqrt{h_{i,j}^4 + \max(h_{i,j}^4, \epsilon)}}.
\end{aligned}
\tag{34}
$$

The determination of ϵ is generally problem specific - we found that setting $\epsilon = 0.01$ provided stable results for a variety of problems. For the sake of consistency, the conserved quantities must be updated with the new velocity values (the water height h can become negative because of numerical round-off errors, so it's clamped to zero here as well):

$$
\begin{aligned}
h_{i,j} &= \max\left(h_{i,j}, 0\right) \\
\boldsymbol{U}_{i,j} &= \begin{pmatrix} h_{i,j} \\ h_{i,j}u_{i,j} \\ h_{i,j}v_{i,j} \end{pmatrix}.
\end{aligned}
\tag{35}
$$

When using the projection to the local characteristic fields before the WENO reconstruction, the definition of the matrix of left eigenvectors becomes ill-posed due to matrices R_F and R_G becoming singular at dry zones:

$$
R_F = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}; \; R_G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}.
\tag{36}
$$

70 We handle this by setting the left and right eigenvector-matrices to the identity matrix at dry zones: this results in reversion to the component-by-component WENO scheme at dry regions. Such a simple treatment is also the likely cause for the previously noted lowered stability of the characteristic projection based scheme near dry regions.

14

3. GPU Implementation

3.1. Choice of Technology

The implementation of the numerical scheme was undertaken using general-purpose GPU (GPGPU) programming. Modern GPUs can handle large amounts of data in parallel and have been shown to be highly effective for real-time fluid simulations [22, 23, 5, 4] - the Eulerian (grid) methodology also maps without complications to the GPU data processing paradigms. As real-time, simultaneous simulation and visualization of the results was paramount to our work, we chose to use the DirectX 12 API's DirectCompute technology (similar to work in [24]). The following discussion however is API agnostic, and GPGPU concepts are generally consistent across different APIs and hardware vendors. Therefore, the discussion is focused on appropriate choice of GPU resources, configuration of the massively parallel program execution as well as our fast boundary condition application methodology. A high-level overview of the final algorithm is also presented.

3.2. Data Storage

Simulation data was stored using texture resources as opposed to buffers for several reasons:

1. Eulerian methods are naturally dependant on spatial discretization properties, hence textures, which are particularly suitable for spatially sampled data [25], are an obvious choice.

2. Modern GPU texture caches provide more flexibility by minimizing the cost of cache misses and unaligned memory accesses [26].

3. Visualization of results is straight-forward as texture coordinates assigned to the visualization mesh can be used to directly query the simulation data.

As GPUs do not support double-precision texture formats, our double precision simulations used two textures with 4 channels of 32bit unsigned integer data

15

each, which were then reinterpreted in pairs as 64bit doubles in the computation kernel. Single precision computations used a texture with 4 channels of
¹⁰⁵ 32bit floating point data. For both setups, the first three components were used for storing the height and momentum terms whereas the last ones were used for encoding the boundary information for our novel boundary-condition application methodology (see Section 3.4.2). As most of the inner domain cells do not contain any meaningful data in the 4th component, one could also use three
¹¹⁰ channel textures and the usual (separate pass) boundary application methods if so desired (this would be preferable when simulation storage requirements are close to the maximum available video memory of the GPU).

3.3. Threading Scheme

Central to any GPGPU implementations is the management of concurrent computation work - the most common method being the decomposition of the simulation domain into fixed size smaller groups (e.g. see [25]). Clearly when performing computations separately in these smaller groups, each of them require their own set of boundary cells in order to correctly process the values at the edges of the group's domain. Henceforth we refer to these boundary cells as local and the ones for the complete domain as global (the number of local boundary cells per dimension is the same as the number of equivalent global boundary cells). The local computation results are written to main memory only for cells that are part of the inner domain in both local as well as global computation domains. Since some grid points are involved in calculations more than once (e.g. once as a local boundary cell whereby its results are discarded and once as a local and global inner domain cell whereby the results are written to main memory), then more threads need to be dispatched than the total number of inner domain cells. The total number of threads to be dispatched for processing all cells can, for example, be found as [27]:

$$\mathcal{D} = \frac{\mathcal{E} + (\mathcal{S} - 1)}{\mathcal{S}} \tag{37}$$

16

where $\mathcal{E}$ is the total number of elements in the dimension to be processed and $\mathcal{S}$ is the number of elements in the smaller group (i.e. the threadgroup; the optimal total size being platform and problem specific) in the given dimension. In order to accommodate for the local boundary cells, $\mathcal{S}$ has to be chosen as follows:

$$\mathcal{S} = \mathcal{Q} - 2\mathcal{B} \tag{38}$$

where $\mathcal{Q} = \{\mathcal{X} \vee \mathcal{Y}\}$ where $\mathcal{X} \times \mathcal{Y}$ is the size of the local threadgroup in x and y
115 directions respectively. $\mathcal{B}$ defines the number of local boundary cells considered per side of a dimension. As an example, consider a scheme that requires 2 boundary cells on each side in the x direction, then $\mathcal{B} = 2$ when $\mathcal{Q} = \mathcal{X}$. An illustration of the described overlapping of threads is shown in Figure 2.
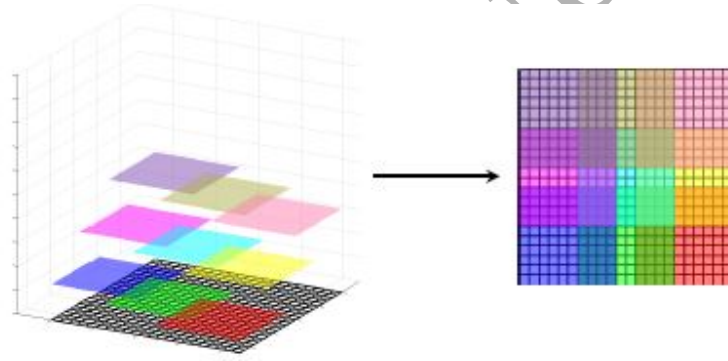

Figure 2: Sample 22×22 grid partitioning. On the left, a 3D view of superimposed local computation grids ($\mathcal{X} = \mathcal{Y} = 10$), on the right a flattened, top-down view of the overlapping grids. When tiling the smaller grids over the larger, it becomes obvious that some cells need to be processed more than once and hence Equations (37) and (38) have to be used for finding the total number of threads to be dispatched for updating all of the inner domain cells.

Indexing into the textures has to be modified accordingly as well. The global indices in the x and y directions can be found as follows:

$$x_g = \mathcal{G}_x \times (\mathcal{X} - 2 \times \mathcal{B}) + \mathcal{T}_x$$
$$y_g = \mathcal{G}_y \times (\mathcal{Y} - 2 \times \mathcal{B}) + \mathcal{T}_y \tag{39}$$

where x_g and y_g are the global indices, the x/y subscripts denote the dimension,

17

$\mathcal{G}$ the group ID and $\mathcal{T}$ the thread ID in the given group. These indices are used only for loading data from main memory to local memory and the final write-back to main memory. All other computations are performed using the thread group local indices. The final writes are governed by the following statements:

$$\mathcal{I}_D = (x_g \geq \mathcal{B}) \wedge (x_g < \mathcal{W} + \mathcal{B}) \wedge (y_g \geq \mathcal{B}) \wedge (y_g < \mathcal{H} + \mathcal{B})$$
$$\mathcal{I}_L = (x_l \geq \mathcal{B}) \wedge (x_l < \mathcal{X} - \mathcal{B}) \wedge (y_l \geq \mathcal{B}) \wedge (y_l < \mathcal{Y} - \mathcal{B}) \qquad (40)$$
$$\text{if } (\mathcal{I}_D \wedge \mathcal{I}_L) \rightarrow \text{store result}$$

where $\mathcal{I}_D$ defines the set of threads that are indexing global inner domain values
120 and $\mathcal{I}_L$ the set that are indexing local inner domain values; $\mathcal{W}$ and $\mathcal{H}$ are the width and height of the global inner domain region; x_l and y_l are the local thread indices.

Additionally, our implementation goal was to avoid splitting the simulation into many separate computation kernels or passes: this follows from the fact that
125 memory reads/writes take a lot longer than arithmetic operations. As Crane *et al.* [23] put it, "math is cheap compared to bandwidth". Vaisse [28] also showed performance gains for their cloth simulation implementations when comparing multi-pass and single pass implementations. We were able to compress the entire scheme into one and two-pass solutions by making heavy use of the above-
130 described local groups setup combined with local synchronization and abstract group shared memory (GSM) usage.

3.4. Boundary Conditions

The purpose of the following discussion is to briefly summarize boundary requirements of the scheme as well as present our novel method for application
135 of boundary conditions within our framework. Note that heavy use is made of DirectX specific behaviour regarding ouf-of-bounds queries in this section (see Appendix B).

3.4.1. Boundary Cell Count

The number of boundary cells required due to our implementation strategy can
140 be found by considering validity of computed data in the GSM of the thread

18

local space. Validity of the data stored in a given GSM cell depends on whether the computation that resulted in that data used well-defined cells. For example, consider the edges of the GSM domain - any central differencing involves values outside the range of the GSM, hence turning the computed result meaningless.

145 Following this idea, the single pass 3rd order PIFWENO scheme requires 4 boundary cells. An example walkthrough of the reasoning as to why this is the case is given in Figure 3 - the necessary boundaries can be derived for a kth order scheme in a similar way. For the two-pass version (split at the time-averaged flux computation stage) one less boundary cell is needed as the stencils in steps
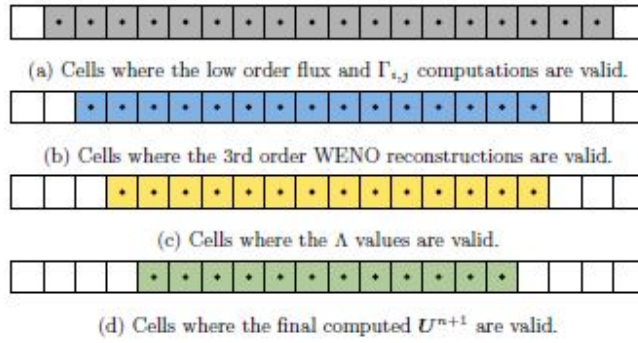
150 a) and b) overlap in this case.

(a) Cells where the low order flux and $\Gamma_{i,j}$ computations are valid.

(b) Cells where the 3rd order WENO reconstructions are valid.

(c) Cells where the Λ values are valid.

(d) Cells where the final computed U^{n+1} are valid.

Figure 3: Boundary cell usage for the PIFWENO3 scheme in the x direction ($\mathcal{X} = 20$; same applies to the y direction).

3.4.2. Boundary Condition Application

For the purposes of our work, we were mainly interested in no-slip boundary conditions, i.e. the velocity in the normal and tangential directions should be zero at the boundaries [29, p.14]. A naïve implementation of boundary

155 conditions inside the main computation kernel would involve branching based on the thread ID, leading to thread divergence and hence potentially reduced performance. As an alternative, we exploited the 4th texture channel - as this component doesn't contain any simulation data, offsets of the boundary cells that would receive a copy (multiplied by a specific boundary vector, e.g. for no-

19

160 slip we can easily deduce $\boldsymbol{B} = (1, -1, -1)^T$) from the currently being processed inner domain cell can be packed into that cell's fourth component. Furthermore, for the 2D SWE, the value of an inner domain cell can be propagated up to a maximum of two possible boundary cell locations (one in x direction and one in y direction). Combining this with the fact that for real time simulations, offsets

165 provided by 16bit integers are sufficient, a total of two 16bit integer offsets can be packed per grid cell into the 4th component of the simulation storage texture. For cells that don't propagate their content to boundary cells, it's sufficient to assign a large offset κ that would direct the writes out-of-bounds given a specific grid configuration. In the program kernel, the values can easily be extracted

170 and stored offsets can be added to the previously computed global indices in order to find the storage ID of the boundary cell. Hence all inner domain cells write their computation results as well as boundary-vector multiplied results into two boundary cells. Three example scenarios are given in Figure 4.
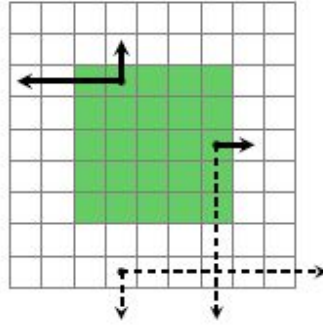

Figure 4: Example scenarios of boundary cell storages ($\mathcal{B} = 2$). The green cells indicate inner domain cells, the solid arrows signify the direction of the packed offsets and the dashed arrows signify offsets that lead to out-of-bounds accesses. At (3, 2) (texture origin at top left), the offsets are -3 in the x direction and -1 in the y direction. On the other hand, at (6, 4), the offsets are +1 and κ and at (3, 8) κ and κ respectively.

Note that this type of boundary condition application is not viable for GSM local

175 periodic boundaries - most other types of boundaries (e.g. in-flow/out-flow) shouldn't pose a problem - as long as the information required by a boundary

20

cell is propagated to it from a nearby cell and not from across the computation domain. Further, it's also important to write the loaded indices back to main memory for the inner domain cells and for the boundary cells being modified

¹⁸⁰ assign a 32bit floating point value which upon unpacking would correspond to indices that lead to out-of-bounds accesses (for example we used the floating point constant 1.99414051f which upon unpacking results in offsets $\kappa_1 = 16383$ and $\kappa_2 = 16383$, see below).

```
i = 0x3fff3fff;        // same constant in hex
k_1 = i >> 16;         // high 16 bits, in dec: 16383
k_2 = (i << 16) >> 16; // low 16 bits, in dec: 16838
```

3.4.3. Flux Boundaries

After computing the time-averaged fluxes, one needs to apply intermediate flux boundaries on the dataset. In our two-pass implementation, these can be applied in the same way as just described (with an appropriate flux boundary vector). Unfortunately, in our one-pass implementation we can't avoid manually handling out-of-bounds writes due to invalidation of shared memory. As these boundary conditions are local, it's sufficient to check only the following condition for setting x directional flux boundaries:

$$\text{if } (x_l + \mathcal{O}_x \geq 0) \wedge (x_l + \mathcal{O}_x < \mathcal{X}) \rightarrow \text{set boundaries} \tag{41}$$

and similarly for the y direction:

$$\text{if } (y_l + \mathcal{O}_y \geq 0) \wedge (y_l + \mathcal{O}_y < \mathcal{Y}) \rightarrow \text{set boundaries} \tag{42}$$

where $\mathcal{O}$ are the unpacked offsets with x/y subscripts denoting direction. As both of these branches involve only a single group-local memory operation then

¹⁹⁰ any thread divergence should cause negligible performance issues.

Note that the flux boundary conditions need to be consistent with the general boundary conditions applied to the simulation domain, i.e. in our case no-slip boundary handling was needed. Considering the fluxes given in the x

21

and y direction in Equation (2), it's clear that for no-slip conditions the time-averaged flux boundary cells need copies of their respective inner domain cell values multiplied by a boundary-vector such that both of the horizontal velocity components are reflected in the resulting flux vector, i.e. the boundary vector for the no-slip flux boundaries is given by $\boldsymbol{B}_{\text{flux}} = (-1, 1, 1)^T$.

3.5. Algorithm Overview

The input resources to the computation kernel(s) are the previous timestep's simulation results and the bathymetry values. The following steps are taken in order to implement the previously described PIFWENO scheme:

1. Load previous timestep's results and extract boundary information.

2. Compute the dissipation coefficient α for the LLF flux splitting, evaluate the fluxes $\boldsymbol{F}(\boldsymbol{U})$ and $\boldsymbol{G}(\boldsymbol{U})$ as well as the respective Jacobian matrices.

3. Compute the Lax-Friedrichs flux for the h component as well as the $(k-1)$th order flux and source term derivatives. Construct the time-averaged fluxes.

4. Set the flux boundary conditions for the x direction and perform the flux splitting. Apply the WENO reconstruction procedure with or without projection to the characteristic fields (for component-by-component versions also absorb the first source term into the flux calculations as suggested by [20]).

5. Repeat previous step for the y direction.

6. Find the limiting θ parameters via the proposed **minimal case** method and construct the final fluxes for the h component.

7. Construct $\boldsymbol{U}^{n+1}$.

8. Desingularize the cell velocities as needed and apply the consistency requirement.

22

220 9. Based on the result of Equation (40), write the results to main memory. Multiply the found U by the boundary vectors and write the results to main memory.

All of these steps were compressed into one and two pass/kernel configurations, the latter of which was constructed due to periodic boundaries. The actual
225 implementation is quite involved, requiring abstract GSM usage and splitting of the computation of equations across local synchronization calls due to the limited available GSM size. A comprehensive description of such a procedure is infeasible and as such we've made the single pass GPU kernels for the 3rd order scheme available online (URL in abstract) with the hope that the previ-
230 ously described strategies combined with the full kernel source codes make any reproduction efforts easier.

4. Results & Discussion

In the following tests we've taken $\epsilon = 0.01$ in Equation (34) and $g = 9.81\text{ms}^{-2}$ unless otherwise stated. All of the results were gathered on a NVIDIA Titan
235 X Pascal GPU. Timing information was obtained using D3D12 Query Heap functionality with the stable power state flag enabled for more consistent results. In the following, we refer to the component by component scheme as PIFWENO3 and the characteristic projection based scheme as PIFWENO3C. Further, we denote single precision computations with SP and double precision
240 computations with DP.

4.1. Validation

Given a parabolic bathymetry and the requirement that the water's surface remain planar throughout the motion, Thacker [30] found an analytical expression for both the surface elevation η and the velocities u and v. This particular test case has been used by several authors [31, 32, 4] before to test the accuracy of their numerical schemes and in general is considered quite a severe test case due to involving shoreline movement [33, 34]. The test case was set up similar to [4],

23

i.e. given a domain $[-3960, 3960] \times [-3960, 3960]$ m^2 with reflective boundaries and a parabolic bottom topography:

$$b = D_0 \left(\frac{x^2 + y^2}{L^2} - 1 \right) \tag{43}$$

where $D_0 = 1$m and $L = 2500$m, the exact solution to the SWE is given by:

$$\eta = \frac{2AD_0}{L^2} \left(x \cos \omega t + y \sin \omega t + LB_0 \right)$$
$$u = -A\omega \sin \omega t \tag{44}$$
$$v = A\omega \cos \omega t$$

where $A = L/2$, $B_0 = -A/2L$ and $\omega = \sqrt{2D_0/L^2}$. Furthermore, the gravitational constant $g = 1$ms^{-2} for this test case. The grid size chosen was 100×100 with $\Delta x = \Delta y = 80$m and a constant timestep $\Delta t = 10$s. The induced motion is periodic with a period of $T = 2\pi/\omega$ - we present slices near the center of the domain (at $y = 40$m) of the simulation at times $t \approx 3T, 3.25T, 3.5T$ for both PIFWENO3 and PIFWENO3C schemes using single and double precision floating point arithmetic in Figures 5-7. The motion within this timespan corresponds to a surface rotation of 7π radians - for a time-lapse from $t = 0$, see the demo video available at the URL given in the abstract.
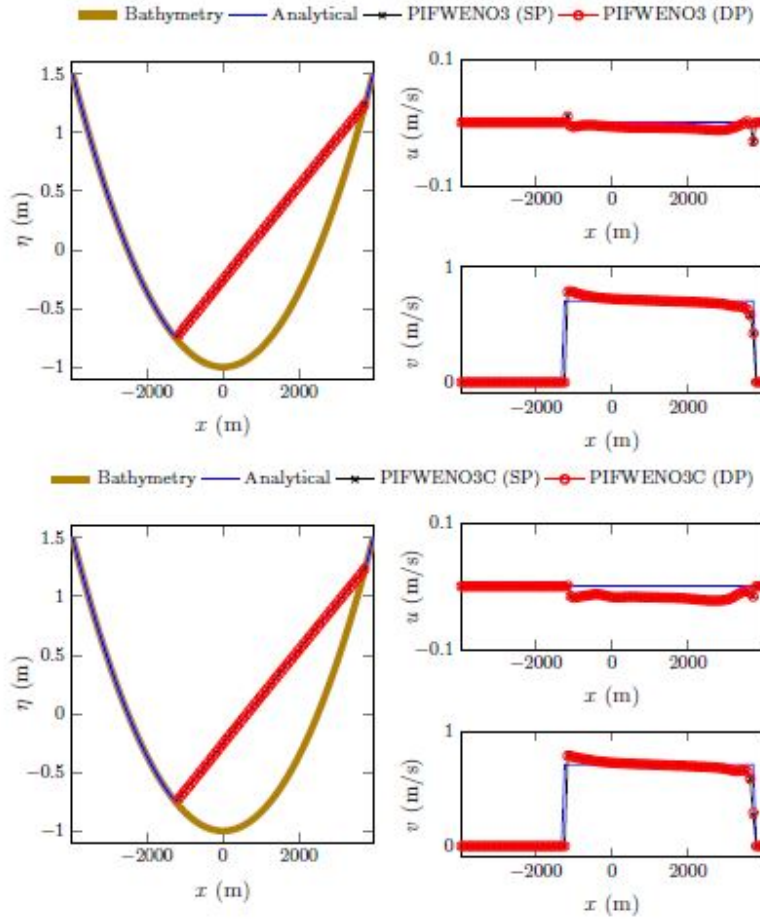
24

Figure 5: PIFWENO3 (top) and PIFWENO3C (bottom) solution plots for η, u and v at $t = 33320\text{s} \approx 3T$.
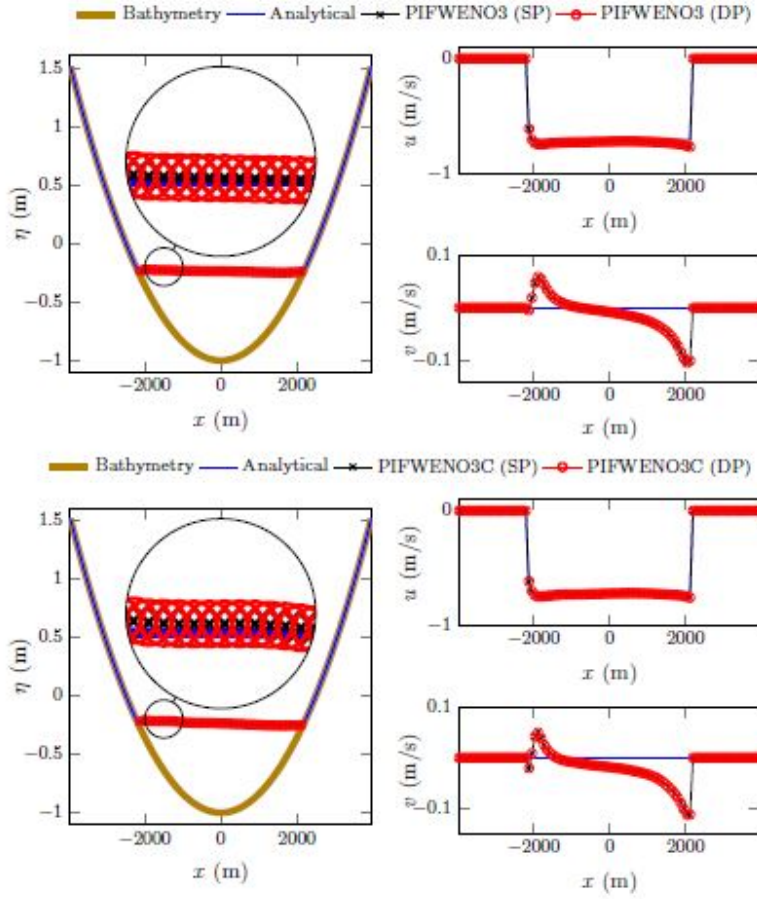
Figure 6: PIFWENO3 (top) and PIFWENO3C (bottom) solution plots for η, u and v at $t = 36100$s $\approx 3.25T$.
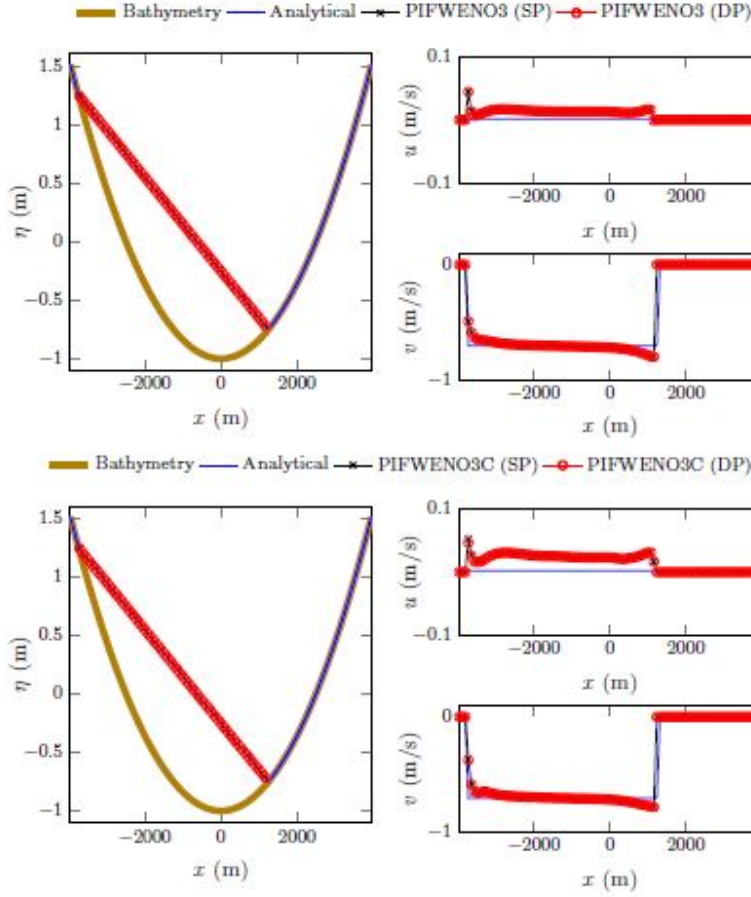
Figure 7: PIFWENO3 (top) and PIFWENO3C (bottom) solution plots for η, u and v at $t = 38880\text{s} \approx 3.5T$.

It's clear from these figures that both schemes provide accurate estimates to the surface elevation η - in Figure 6 one can however notice some small disturbances in the PIFWENO3C surface in the highlighted $x < 0$ region. On the other hand, the velocity profiles show much larger errors for both schemes - this is to be expected, as an accurate simulation of the velocity profile for this test case is known to be problematic [33, 35, 36] and the gathered results are consistent with those in other works [31, 34, 37]. The differences between the

27

characteristic and component based schemes are especially apparent in Figures 5 and 7 for the u components. Interestingly, double precision does not seem to have a major effect on either the accuracy of the surface elevation nor velocity profiles - this is potentially due to the fact that the velocity desingularization at dry zones requires square root operations which HLSL does not support for double precision [38, 39]. In order to investigate the discrepancy between the accuracy of the schemes, Figure 8 shows the evolution of the average error over the entire simulation grid for both methods over three periods of motion. Evidently, the error profiles for η, u and v have very similar structures with the characteristic based scheme performing worse than the component based one. The ill-posed nature of the characteristic based scheme at dry zones clearly has a negative effect on the solution - future work could look at alternatives to the reversion to the component based scheme. The differences between single and double precision results were once again found to be small. Note that while the error profiles for the velocity components are highly oscillatory for both of the schemes, the actual resulting velocity fields retain their overall smoothness and hence the resulting circular surface also retains its shape despite the arisen errors.
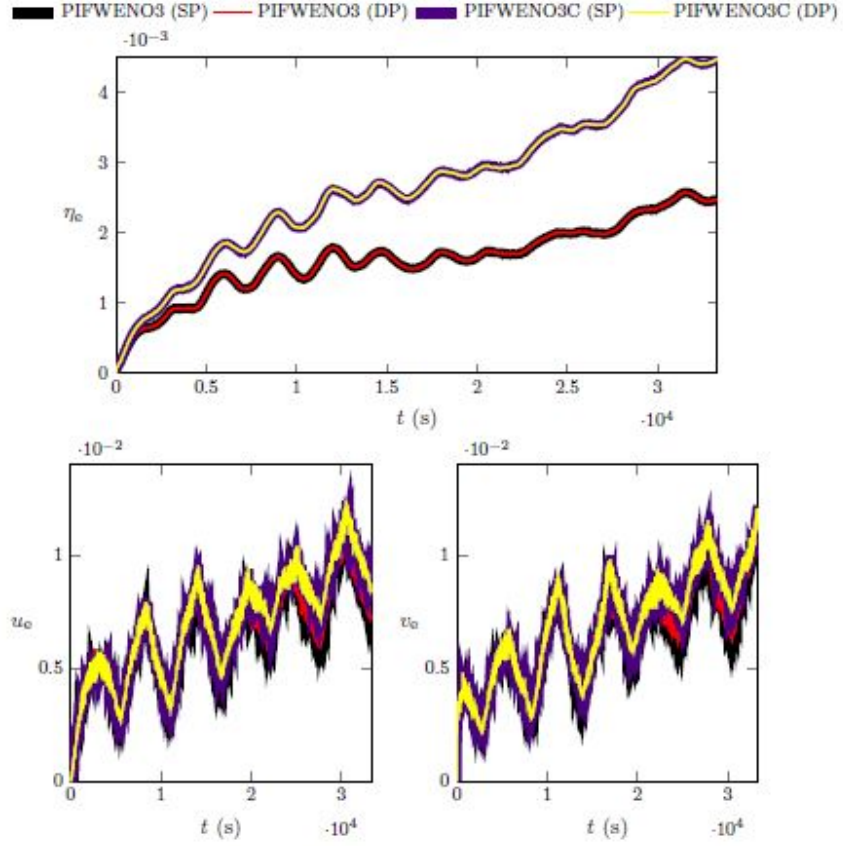
Figure 8: PIFWENO3 and PIFWENO3C average errors over 3 periods of motion for η, u and v.

4.2. Well-Balanced (C-property) Test

The next test was designed to check the well-balanced (C-property) of the presented schemes. The bathymetry and initial conditions for this test were chosen as:

$$b(x,y) = \begin{cases} 0.8 & \text{if } x > 0.8 \\ 0.5 \sin\left(4\pi x\right)\cos\left(4\pi y\right) & \text{otherwise} \end{cases} \tag{45}$$

$$\boldsymbol{U}(x,y,0) = \begin{pmatrix} 1 - b(x,y) & 0 & 0 \end{pmatrix}^T$$

29

on a domain $[0.005, 0.995] \times [0.005, 0.995]$ m^2 with reflective boundaries. Further, $\Delta x = \Delta y = 0.01$m and $\Delta t = (\Delta x/20)$s. The grid size for the test was 100×100 and the end time was $t \approx 0.2$s. Note that the bathymetry has both smooth and discontinuous components. **In general, the developed schemes were found to handle discontinuities within the bathymetry without issues when dealing with completely wet domains. On the other hand, instabilities were observed for some scenarios involving wetting and drying processes (more so for the characteristic variations). For our purposes, slightly smoothing the bathymetry in a pre-processing stage for those scenarios was found to alleviate any arising issues and hence further analysis was not performed.** The L_1 and L_∞ errors (for more info see e.g. [40, Appendix A]) using single and double precision are presented in Table 1 - the C-property is confirmed by the fact that both schemes reached machine-precision level errors in their respective calculations. Note that similar to the scheme by Kurganov and Petrova [8], neither of the PIFWENO schemes maintain this C-property when the bathymetry pierces the surface. In our simulations we made observations similar to those in [4] - these small flux imbalances have a negligible effect on the overall solution and the results were found to be satisfactory for our purposes. Future work could look at further modifying the flux calculations to allow for accurate steady-state solutions with surface-piercing bathymetry functions.

Norm	PIFWENO3			PIFWENO3C		
	h	hu	hv	h	hu	hv
L_1	4.38e-08	3.11e-07	2.68e-07	7.18e-08	4.08e-07	4.01e-07
	3.66e-17	5.12e-16	4.77e-16	1.82e-16	7.53e-16	6.93e-16
L_∞	3.28e-07	1.99e-06	1.94e-06	5.07e-07	2.17e-06	2.59e-06
	4.44e-16	3.01e-15	3.24e-15	1.28e-15	5.01e-15	4.67e-15

Table 1: Well-balanced test results for PIFWENO3 and PIFWENO3C schemes.

30

4.3. Grid Convergence

The purpose of the following test was to verify that the practical convergence rate of the scheme matches its theoretical order of accuracy for smooth solutions. To this end, we used a similar setup to [20], i.e. the initial conditions were defined as:

$$b(x, y) = \sin(2\pi x) + \cos(2\pi y)$$

$$\boldsymbol{U}(x, y, 0) = \begin{pmatrix} 10 + \exp(\sin(2\pi x))\cos(2\pi y) \\ \sin(\cos(2\pi x))\sin(2\pi y) \\ \cos(2\pi x)\cos(\sin(2\pi y)) \end{pmatrix} \tag{46}$$

on a domain $[0, 1) \times [0, 1)$ m^2 with periodic boundaries (hence we used a modified version of our 2-pass implementation for this test case (see below)). Additionally, we set $\Delta x = \Delta y = (1/N)$m where N is the number of grid cells per dimension - the fine grid solution at 1600×1600 was considered the reference solution and hence the computations were run on grids with $N \in \{25, 50, \ldots, 1600\}$. The end time was chosen as $t = 0.05$s. In order to minimize the temporal error, the tests were run with a CFL number $\nu = 0.2$, defined as [13, p.70] [41]:

$$\nu = \Delta t \left(\frac{u_{\max}}{\Delta x} + \frac{v_{\max}}{\Delta y} \right) \tag{47}$$

where $u_{\max}$ and $v_{\max}$ are the maximum eigenvalues in the x and y directions, respectively. As the previously described GPU implementation was designed for constant timestepping, additional passes had to be written for enforcing the CFL condition: a single pass over the entire computation grid was used to find maximum eigenvalues per grid cell, followed by basic parallel reduction in GSM for finding the maximum values for each thread group. A further additional pass was then used to find the global maximum eigenvalues and hence the timestep size - this brought the number of total computation passes up to 4 for this test case. Note that when dealing with very large grids, it might be beneficial to introduce additional GSM-based parallel reduction passes in order to speed up the global maximum computation (due to a limited maximum number of allowed threads per group, it might not be possible to reduce the number of maximum eigenvalues to a sufficiently small number in just a single pass).

31

The convergence test results are given in Tables 2 and 3 for PIFWENO3 and PIFWENO3C schemes, respectively. It's clear that with grid refinement, the computed order approaches the expected value of 3 (i.e. the spatial order) for
315 both schemes. We note that our Taylor timestepping solution does not reach hyper-convergence levels as often seen with Runge-Kutta method-of-lines solvers (cf. [11]).

Mesh	h		hu		hv	
	L_1	Order	L_1	Order	L_1	Order
25^2	7.38e-02		4.21e-01		8.67e-01	
50^2	2.03e-02	1.86	1.09e-01	1.96	1.81e-01	2.26
100^2	4.26e-03	2.25	2.18e-02	2.32	3.38e-02	2.42
200^2	8.55e-04	2.32	4.17e-03	2.39	6.83e-03	2.31
400^2	1.76e-04	2.28	8.77e-04	2.25	1.45e-03	2.24
800^2	3.33e-05	2.40	1.54e-04	2.51	2.93e-04	2.30

Table 2: Grid convergence test results for PIFWENO3.

Mesh	h		hu		hv	
	L_1	Order	L_1	Order	L_1	Order
25^2	7.78e-02		4.15e-01		9.27e-01	
50^2	2.53e-02	1.62	1.12e-01	1.89	2.20e-01	2.08
100^2	6.01e-03	2.07	2.66e-02	2.07	4.85e-02	2.18
200^2	1.16e-03	2.37	5.20e-03	2.35	8.70e-03	2.48
400^2	2.03e-04	2.51	8.82e-04	2.56	1.53e-03	2.51
800^2	3.34e-05	2.60	1.35e-04	2.71	2.79e-04	2.46

Table 3: Grid convergence test results for PIFWENO3C.

32

4.4. Complex Heightfield Test

The purpose of this last test was to investigate the ability of the schemes to model complex flow features with generic bathymetry data. To achieve this, we used L3DT [42] to generate a detailed heightmap dataset which we then applied over the following initial conditions:

$$b(x,y) = \max\left(0, 4\sin\left(\frac{\pi x}{24}\right)\right) + \mathcal{H}(x,y)$$

$$\Psi = \max\left(0, \cos\left(\frac{\pi}{2} + \frac{2\pi x}{24}\right)\right) + \begin{cases} \max\left(0, 1.5 - b(x,y)\right) & \text{if } x < 3 \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{U}(x,y,0) = \begin{pmatrix} \Psi & 0 & 0 \end{pmatrix}^T$$

$$\tag{48}$$

where $\mathcal{H}$ is the value at (x,y) in the heightfield dataset. The test was set up on a
320   1600×1152 grid over a numerical domain defined in $(-12, 12) \times (-8.75, 8.75)$. We used a slightly modified $\Delta x = \Delta y = 0.2$m and $\Delta t = 0.016$s for our simulations in order to approximately synchronize the simulation to 60 frames per second (FPS) rendering. 3D snapshots and the corresponding contour plots given in Figure 9 show that the PIFWENO3 scheme can clearly handle complicated
325   flows without issues (PIFWENO3C results omitted for brevity as they were very similar). A video showing real-time as well as faster than real-time simulation results from this test case can be found online (one can also find the heightfield dataset there, URL in abstract).
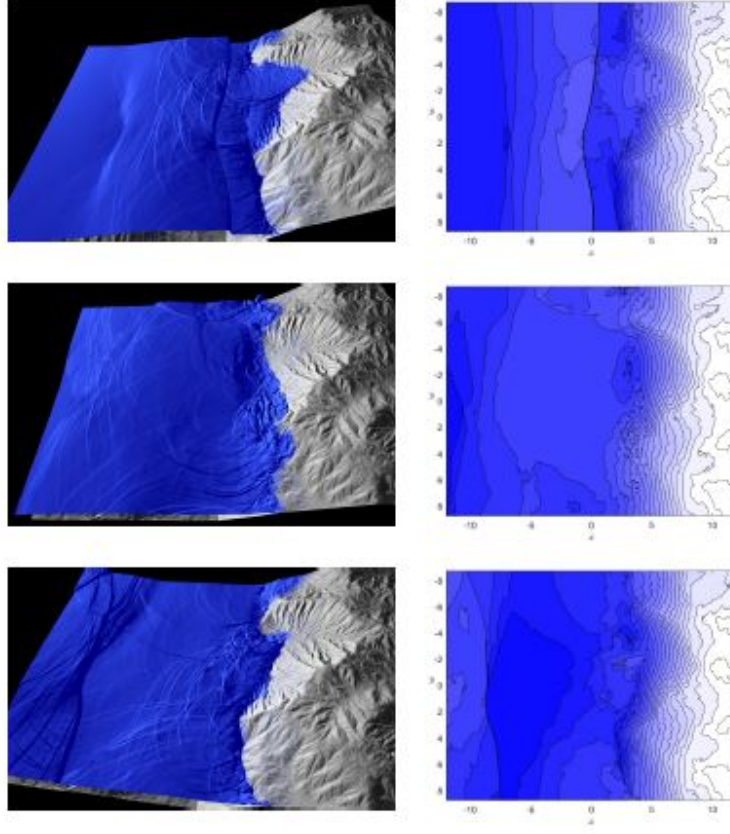
33

Figure 9: Complex heightfield test for the PIFWENO3 scheme at $t \approx 50, 70, 90s$. Left - 3D surface renders, right - **surface elevation** η contour plots.

4.5. Performance Evaluation

The purpose of the following discussion is to assess the suitability of the presented schemes for real-time simulation and rendering. A brief overview of the single pass computation kernels for both single and double precision is given in Table 4 - the threadgroup sizes were found through thorough testing of various different configurations, aided by the CUDA Occupancy Calculator [43]. Due to a number of different conflicting optimization parameters (for a discussion see e.g. [4]), optimizing for maximum occupancy was not always found to produce best results. Further, even the common practice of keeping the number of

34

threads to an integer multiple of the warp size was found to not always yield expected results (e.g. PIFWENO3C in double precision was found to perform best with a non-integer number of warps per thread group).

	PIFWENO3		PIFWENO3C	
	SP	DP	SP	DP
Registers	20	34	37	52
Threads ($\mathcal{X} \times \mathcal{Y}$)	32×24	24×16	24×24	20×20
GSM (bytes)	30720	30720	23040	32000
Instructions	390	843	564	1375
Occupancy (%)	75	38	56	41

Table 4: Overview of the computation kernels for the single pass schemes.

In theory, PIFWENO3 in single precision should perform the best - the kernel has significantly less instructions and uses far fewer registers than the other kernels. In order to verify this, the performance of the schemes was measured on varying grid sizes - as all cells were treated equal in the schemes, the results were found to be independent of the setup of the problem and hence we present results for a slightly modified ($g = 9.81\text{ms}^{-2}$, $\Delta x = \Delta y = (1/N)\text{m}$, $\Delta t = (\Delta x/20)\text{s}$) version of the validation test case (Section 4.1). Measured timings for a single timestep over an average of 1000 iterations are given in Table 5.

35

	PIFWENO3		PIFWENO3C	
Mesh	SP	DP	SP	DP
64^2	0.015	0.414 ($\uparrow 27.6\times$)	0.018 ($\uparrow 1.2\times$)	0.854 ($\uparrow$ 56.9$\times$)
128^2	0.016	1.043 ($\uparrow 65.2\times$)	0.027 ($\uparrow 1.7\times$)	1.518 ($\uparrow$ 94.9$\times$)
256^2	0.040	2.509 ($\uparrow 62.7\times$)	0.061 ($\uparrow 1.5\times$)	3.579 ($\uparrow$ 89.5$\times$)
512^2	0.141	7.533 ($\uparrow 53.4\times$)	0.223 ($\uparrow 1.6\times$)	11.882 ($\uparrow$ 84.3$\times$)
1024^2	0.358	29.849 ($\uparrow 83.4\times$)	0.630 ($\uparrow 1.8\times$)	46.913 ($\uparrow 131.0\times$)
2048^2	1.374	116.020 ($\uparrow 84.4\times$)	2.461 ($\uparrow 1.8\times$)	179.280 ($\uparrow 130.5\times$)
4096^2	5.628	455.518 ($\uparrow 80.9\times$)	10.176 ($\uparrow 1.8\times$)	696.508 ($\uparrow 123.8\times$)

Table 5: Computational performance of the schemes. Timings in milliseconds, the value in brackets signifies the change in computation time compared to the single precision PIFWENO3 kernel.

From Table 5, it's clear that the single precision kernels run much faster on the GPU - while double precision support for GPUs has grown significantly, it is not a priority in consumer GPUs such as the one used in this paper. Since double precision results were comparable to single precision results for the validation test case, we recommend first experimenting with single precision before using the considerably more expensive double precision arithmetic. Single precision is also the only viable option for applications requiring simultaneous simulation and rendering of the results in real-time - for a smooth 60FPS ($\approx$16.67ms) rendering, one actually encounters drops in framerate due to the amount of geometry required for rendering both the bathymetry and water surfaces. In contrast, double precision computations on grids $> 512^2$ already exceed the 16ms threshold by themselves. The only test case of the ones we've presented that really required double precision was the grid convergence test. The table also shows that the characteristic projection based scheme performs, on average, $\sim$1.6$\times$ worse than the per-component version. As this extra computational expense does not come with better accuracy (as shown by the validation test), we find the component based version to be the better option of the two - at

36

least for simulations involving dry zones.

In order to investigate our hypothesis that fewer simulation passes result in better overall performance, the same performance test was repeated with a two-pass version needed for handling periodic boundary conditions. As both the single and double pass implementations shared the same code-paths and hence resources, the two-pass kernels in this test also used 4 boundary cells and the same threading configuration for both passes. The test results for both of the schemes using the faster single precision kernels are given in Table 6. This showcases an average of $\sim 1.4\times$ higher performance for single pass kernels. The performance gap could potentially be lowered by different threading schemes for either of the passes, however we do not see a reason for further optimizing the 2-pass variation when it's clear that the 1-pass version is, if not faster, at least comparative in speed to the 2-pass version.

	PIFWENO3 (SP)		PIFWENO3C (SP)	
Mesh	1-pass	2-pass	1-pass	2-pass
64^2	0.015	0.022 ($\uparrow 1.5\times$)	0.018	0.026 ($\uparrow 1.4\times$)
128^2	0.016	0.027 ($\uparrow 1.7\times$)	0.027	0.038 ($\uparrow 1.4\times$)
256^2	0.040	0.057 ($\uparrow 1.4\times$)	0.061	0.082 ($\uparrow 1.3\times$)
512^2	0.141	0.199 ($\uparrow 1.4\times$)	0.223	0.320 ($\uparrow 1.4\times$)
1024^2	0.358	0.478 ($\uparrow 1.3\times$)	0.630	0.782 ($\uparrow 1.2\times$)
2048^2	1.374	1.829 ($\uparrow 1.3\times$)	2.461	3.034 ($\uparrow 1.2\times$)
4096^2	5.628	7.820 ($\uparrow 1.4\times$)	10.176	12.553 ($\uparrow 1.2\times$)

Table 6: Computational performance of the one and two-pass versions of the schemes. Timings in milliseconds, the value in brackets signifies the change in computation time compared to the relevant single pass kernel.

4.6. Increasing the Orders of Accuracy

As mentioned in the introduction, one of the benefits of employing the WENO reconstructions in a numerical method is it's capability to be extended to arbi-

37

trary orders. In this paper we've focused on a third order in space and second order in time solution, however we have also done early experiments with higher order formulations and think the following information might be of interest to ₃₈₅ those using e.g. the classic 5th order WENO reconstructions:

- We've found the 5th order WENO to require at least 3rd order timestepping to achieve grid convergence. The need for higher order timestepping is further suggested by initial comparisons of the Thacker test case whereby PIFWENO5 variations using just second order timestepping result in loss ₃₉₀ of accuracy over the PIFWENO3 schemes. However, note that higher order tensors involved in the computation of the time averaged fluxes include terms involving $1/h$ which further complicate the simulation in regions where $h = 0$.

- The ratio of local inner domain cells to local boundary cells is reduced ₃₉₅ significantly with higher orders (e.g. 5th order requires 6 boundary cells for the single pass). This results in a lot more overlapping cells and hence more work to be done by the GPU on top of the more involved reconstruction procedures.

- Higher order formulations on current hardware will likely be register bound ₄₀₀ - even PIFWENO3C in double precision with the 20×20 configuration uses more registers than recommended by the HLSL compiler. Therefore, multi-pass solutions could potentially give better performance in these situations (at least on current hardware; see also [44]).

5. Conclusions

₄₀₅ We have presented a new high order finite difference based numerical method for solving the 2D shallow water equations and showcased the numerical accuracy as well as computational performance of both the component by component and characteristic projection based variations. For simulations involving moving shorelines, the presented results indicate higher accuracy for the component

38

based scheme. This can be explained by the simplistic approach taken to handling the ill-posed nature of the projection matrices when $h = 0$ - future work could look at alternative, smoother transitions. **We also presented a reformulated GPU-friendly positivity preservation method alongside a novel fast boundary condition application technique**. Combining these developments with the compact nature of the PIFWENO formulation, we were able to implement the entire numerical scheme in a single GPU kernel and as a result achieved real-time simultaneous simulation and rendering using single precision arithmetic even on large ($> 2000 \times 2000$) grids. Comparisons to double precision computations showcased negligible accuracy gains with significant performance loss (more than $50\times$) and as such we recommend experimenting with single precision calculations before choosing the more expensive double precision route. Based on the presented results, we recommend the single precision component based scheme for those who are interested in real-time simultaneous simulation and rendering of the 2D SWE - the GPU friendly formulation has greatly benefited our work as even very large (4096×4096) grids can be processed using just 5.6ms of GPU compute time per timestep on consumer-level hardware. Future work could look at modelling frictional forces similar to [4], well-balanced treatment of the wet-dry interface or further increasing temporal/spatial orders of the scheme. We have provided a brief discussion of preliminary results and pointed out issues pertaining to the final item for those interested in taking the scheme further.

References

[1] D. J. Acheson, Elementary Fluid Dynamics, Oxford University Press, 1990.

[2] D. Che, L. W. Mays, Development of an Optimization/Simulation Model for Real-Time Flood-Control Operation of River-Reservoirs Systems, Water Resources Management 29 (11) (2015) 3987–4005. `doi:10.1007/s11269-015-1041-8`.

[3] W.-Y. Liang, T.-J. Hsieh, M. T. Satria, Y.-L. Chang, J.-P. Fang, C.-C. Chen, C.-C. Han, A GPU-Based Simulation of Tsunami Propagation and Inundation, Springer Berlin Heidelberg, 2009, pp. 593–603. `doi:10.1007/978-3-642-03095-6_56`.

[4] A. R. Brodtkorb, M. L. Sætra, M. Altinakar, Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation, Computers and Fluids 55 (2012) 1–12. `doi:10.1016/j.compfluid.2011.10.012`.

[5] N. Chentanez, M. Müller, Real-time Simulation of Large Bodies of Water with Small Scale Details, in: Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 2010.

[6] T.-J. Hsieh, W.-Y. Liang, Y.-L. Chang, M. T. Satria, B. Huang, Parallel tsunami simulation and visualization on tiled display wall using OpenGL Shading Language, Journal of the Chinese Institute of Engineers 36 (2) (2013) 202–211. `doi:10.1080/02533839.2012.727606`.

[7] O. T. Ransom, B. A. Younis, Explicit GPU Based Second-Order Finite-Difference Modeling on a High Resolution Surface, Feather River, California, Water Resources Management 30 (1) (2016) 261–277. `doi:10.1007/s11269-015-1160-2`.

[8] A. Kurganov, G. Petrova, A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system, Communications

40

465   in Mathematical Sciences 5 (1) (2007) 133–160. `doi:10.4310/CMS.2007.`
      `v5.n1.a6.`

[9] A. Bollermann, G. Chen, A. Kurganov, S. Noelle, A Well-Balanced Reconstruction of Wet/Dry Fronts for the Shallow Water Equations, Journal of Scientific Computing 56 (2) (2013) 267–290. `doi:10.1007/`
470   `s10915-012-9677-5.`

[10] G.-S. Jiang, C.-W. Shu, Efficient Implementation of Weighted ENO Schemes, Journal of Computational Physics 126 (1996) 202–228.

[11] D. C. Seal, Y. Güçlü, A. J. Christlieb, The Picard integral formulation of weighted essentially non-oscillatory schemes (2014) 1–24.
475   URL `https://arxiv.org/abs/1403.1282v2`

[12] D. C. Seal, Q. Tang, Z. Xu, A. J. Christlieb, An Explicit High-Order Single-Stage Single-Step Positivity-Preserving Finite Difference WENO Method for the Compressible Euler Equations, Journal of Scientific Computing 68 (1) (2016) 171–190. `doi:10.1007/s10915-015-0134-0.`

480 [13] R. J. LeVeque, Finite-Volume Methods for Hyperbolic Problems, Cambridge University Press, 2002.

[14] A. Harten, B. Engquist, S. Osher, S. R. Chakravarthy, Uniformly high order accurate essentially non-oscillatory schemes, III, Journal of Computational Physics 71 (2) (1987) 231–303. `doi:10.1016/0021-9991(87)90031-3.`

485 [15] C.-W. Shu, Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws, Tech. rep. (1997).

[16] C.-W. Shu, Lecture 2: Finite Difference WENO Schemes (2013).
      URL   `http://dauns.math.tulane.edu/~kurganov/`
      `CliffordLectures2013/Shu2.pdf`

490 [17] N. Črnjarić-Žic, S. Vuković, L. Sopta, On different flux splittings and flux functions in WENO schemes for balance laws, Computers & Fluids 35 (10) (2006) 1074–1092. `doi:10.1016/j.compfluid.2005.08.005.`

41

[18] G.-S. Jiang, C.-c. Wu, A High-Order WENO Finite Difference Scheme for the Equations of Ideal Magnetohydrodynamics, Journal of Computational Physics 150 (2) (1999) 561–594. doi:10.1006/jcph.1999.6207.

[19] C. Lu, G. Li, Simulations of Shallow Water Equations by Finite Difference WENO Schemes with Multilevel Time Discretization, Numerical Mathematics: Theory, Methods and Applications 4 (4) (2011) 505–524. doi:10.4208/nmtma.2011.m1027.

[20] Y. Xing, C.-W. Shu, High order finite difference WENO schemes with the exact conservation property for the shallow water equations, Journal of Computational Physics 208 (1) (2005) 206–227. doi:10.1016/j.jcp.2005.02.006.

[21] C. Liang, Z. Xu, Parametrized Maximum Principle Preserving Flux Limiters for High Order Schemes Solving Multi-Dimensional Scalar Hyperbolic Conservation Laws, Journal of Scientific Computing 58 (2014) 41–60. doi:10.1007/s10915-013-9724-x.

[22] M. J. Harris, Fast Fluid Dynamics Simulation on the GPU, in: R. Fernando (Ed.), GPU Gems, 2004, Ch. 38.

[23] K. Crane, I. Llamas, S. Tariq, Real-Time Simulation and Rendering of 3D Fluids, in: H. Nguyen (Ed.), GPU Gems 3, 2007, Ch. 30.

[24] R. Falconer, A. Houston, Visual Simulation of Soil-Microbial System Using GPGPU Technology, Computation 3 (1) (2015) 58–71. doi:10.3390/computation3010058.

[25] J. Zink, M. Pettineo, J. Hoxley, Practial Rendering and Computation with Direct3D 11, CRC Press, 2011.

[26] L. Nyland, S. Jones, Inside Kepler (2012).
URL http://on-demand.gputechconf.com/gtc/2012/presentations/S0642-GTC2012-Inside-Kepler.pdf

42

[27] J. Sanders, E. Kandrot, CUDA By Example: an introduction to general-purpose GPU programming, Addison-Wesley, 2010.

[28] A. Vaisse, Efficient usage of compute shaders on Xbox One and PS4 (2014).
URL http://twvideo01.ubm-us.net/o1/vault/gdceurope2014/Presentations/828884_Alexis_Vaisse.pdf

[29] R. Bridson, Fluid Simulation for Computer Graphics, A K Peters Series, Taylor & Francis, 2008.

[30] W. C. Thacker, Some exact solutions to the nonlinear shallow-water wave equations, Journal of Fluid Mechanics 107 (1981) 499–508. doi:10.1017/S0022112081001882.

[31] R. Holdahl, H. Holden, K.-A. Lie, Unconditionally Stable Splitting Methods for the Shallow Water Equations, BIT Numerical Mathematics 39 (3) (1999) 451–472. doi:10.1023/A:1022366502335.

[32] O. Delestre, C. Lucas, P.-A. Ksinant, F. Darboux, C. Laguerre, T. N. T. Vo, F. James, S. Cordier, SWASHES: a compilation of Shallow Water Analytic Solutions for Hydraulic and Environmental Studies, International Journal for Numerical Methods in Fluids 72 (3) (2011) 269–300. doi:10.1002/fld.3741.

[33] M. E. Hubbard, N. Dodd, A 2D numerical model of wave run-up and overtopping, Coastal Engineering 47 (1) (2002) 1–26. doi:10.1016/S0378-3839(02)00094-7.

[34] A. I. Delis, I. K. Nikolos, M. Kazolea, Performance and Comparison of Cell-Centered and Node-Centered Unstructured Finite Volume Discretizations forShallow Water Free Surface Flows, Archives of Computational Methods in Engineering 18 (1) (2011) 57–118. doi:10.1007/s11831-011-9057-6.

[35] A. I. Delis, I. K. Nikolos, A novel multidimensional solution reconstruction and edge-based limiting procedure for unstructured cell-centered fi-

43

nite volumes with application to shallow water dynamics, International Journal for Numerical Methods in Fluids 71 (5) (2013) 584–633. `doi: 10.1002/fld.3674`.

[36] Z. Horváth, J. Waser, R. A. P. Perdigão, A. Konev, G. Blöschl, A Two-Dimensional Numerical Scheme of Dry/Wet Fronts for the Saint-Venant System of Shallow Water Equations, International Journal for Numerical Methods in Fluids 77 (3) (2015) 159–182. `doi:10.1002/fld.3983`.

[37] J. Hou, Q. Liang, H. Zhang, R. Hinkelmann, An efficient unstructured MUSCL scheme for solving the 2D shallow water equations, Environmental Modelling & Software 66 (2015) 131–152. `doi:10.1016/j.envsoft.2014.12.007`.

[38] Microsoft, D3D12_FEATURE_DATA_D3D12_OPTIONS structure (Windows).
URL `https://msdn.microsoft.com/en-us/library/windows/desktop/dn770364(v=vs.85).aspx`

[39] Microsoft, D3D11_FEATURE_DATA_D3D11_OPTIONS structure (Windows).
URL `https://msdn.microsoft.com/en-us/library/windows/desktop/hh404457(v=vs.85).aspx`

[40] R. J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems, SIAM, 2007.

[41] CFD-Online, Courant-Friedrichs-Lewy condition – CFD-Wiki, the free CFD reference.
URL `https://www.cfd-online.com/Wiki/Courant%E2%80%93Friedrichs%E2%80%93Lewy_condition`

[42] A. Torpy, L3DT (2016).
URL `http://www.bundysoft.com/L3DT/downloads/standard.php`

44

[43] NVIDIA, CUDA Occupancy Calculator (2016).
URL   http://developer.download.nvidia.com/compute/cuda/CUDA_
Occupancy_calculator.xls

[44] J. Pekkilä, M. Väisälä, M. J. Käpylä, P. J. Käpylä, O. Anjum, Methods for
compressible fluid simulation on GPUs using high-order finite differences,
Computer Physics Communications 217 (2017) 11–22. doi:10.1016/j.
cpc.2017.03.011.

[45] C. Gardner, WENO-LF.
URL https://math.la.asu.edu/~gardner/weno.pdf

[46] F. A. Andiga, A. Baeza, A. M. Belda, P. Mulet, Analysis of WENO Schemes
for Full and Global Accuracy, SIAM J. Numer. Anal. 49 (2) (2011) 893–915.
doi:10.1137/100791579.

[47] I. Cravero, M. Semplice, On the Accuracy of WENO and CWENO Re-
constructions of Third Order on Nonuniform Meshes, Journal of Scientific
Computing 67 (3) (2016) 1219–1246. doi:10.1007/s10915-015-0123-3.

[48] C. Boyd, DirectX 11 Compute Shader (2008).
URL http://s08.idav.ucdavis.edu/boyd-dx11-compute-shader.pdf

[49] Microsoft, ld_structured (sm5 - asm) (2016).
URL https://msdn.microsoft.com/en-us/library/windows/desktop/
hh447157(v=vs.85).aspx

[50] Microsoft, store_structured (sm5 - asm) (2016).
URL https://msdn.microsoft.com/en-us/library/windows/desktop/
hh447237(v=vs.85).aspx

Appendix A. WENO3 Reconstruction

A single WENO reconstruction step for a value v involves the following (the
$\pm$ superscripts denote which bias the stencil uses, x direction as an example, j
offsets dropped for convenience):

45

1. Find the candidate stencil approximations. For the third order scheme ($k = 3$, $r = 2$) these are [45]:

$$
\begin{aligned}
f_0^+ &= \tfrac{1}{2}v_i + \tfrac{1}{2}v_{i+1} & f_0^- &= \tfrac{1}{2}v_{i+1} + \tfrac{1}{2}v_i \\
f_1^+ &= -\tfrac{1}{2}v_{i-1} + \tfrac{3}{2}v_i & f_1^- &= -\tfrac{1}{2}v_{i+2} + \tfrac{3}{2}v_{i+1}.
\end{aligned}
\tag{A.1}
$$

2. Find the smoothness indicators using the method by [10]. For the third order scheme the explicit equations are [45]:

$$
\begin{aligned}
\beta_0^+ &= (v_{i+1} - v_i)^2 & \beta_0^- &= (v_{i+1} - v_i)^2 \\
\beta_1^+ &= (v_i - v_{i-1})^2 & \beta_1^- &= (v_{i+2} - v_{i+1})^2.
\end{aligned}
\tag{A.2}
$$

3. Find the non-linear weights as [15, p.18]:

$$
\omega_n^\pm = \frac{\zeta_n^\pm}{\sum_{m=0}^{r-1} \zeta_m^\pm}
\tag{A.3}
$$

where $n \in [0, r-1]$. Furthermore,

$$
\zeta_n^\pm = \frac{d_n}{\left(\epsilon + \beta_n^\pm\right)^2}
\tag{A.4}
$$

where d_n are the linear weights and $\epsilon > 0$ which was originally chosen in order to avoid the denominator becoming zero. Recently, it's been shown that ϵ should actually be related to the grid-spacing or the square of the grid-spacing [46, 47] - hence in this work we've used $\epsilon = (\Delta x)^2$. The d_n for the third order scheme are $d_0 = \tfrac{2}{3}$ and $d_1 = \tfrac{1}{3}$.

4. Find the kth order accurate approximation [15, p.18]:

$$
v_{i+1/2}^\pm = \sum_{n=0}^{r-1} \omega_n^\pm f_n^\pm.
\tag{A.5}
$$

Appendix B. Out-of-bounds Accesses

Our implentation heavily relies on the fact that there's no explicit need to check for out-of-bounds reads when using DirectX. Out-of-bounds reads from main memory return zeros for all components [48] whereas the return results from

46

groupshared memory are undefined [49] - note that this doesn't cause GSM invalidation. In the latter case, it is often known *a priori* that the data received from out-of-bounds queries would not be used further in the computations and hence explicit branching is omitted. Writes to out-of-bounds main memory locations are guaranteed to be no-ops by the DirectX API [48]. No out-of-bounds writes to local memory take place as such actions would cause invalidation of the entire GSM [50]. We highly recommend verifying these assumptions if any other API is used - if unsure, protective branches should be used.