Interactive Boussinesq-type Simulation and Visualization of Water Wave Propagation on GPU

Sasan Tavakkol, Patrick Lynett, Aiichiro Nakano

University of Southern California

1. Abstract

A coastal wave simulation and visualization software is developed based on the Boussinesq equations. Both simulation and its concurrent visualization are performed on the GPU using DirectX API. The software provides faster than real-time, interactive modeling for the first time in coastal engineering. A model running faster than real-time can be handy in disaster forecasting, naval navigation, and any time-sensitive project. Interactivity provides the option for scientists and engineers to test different scenarios and see the results on the go. To improve the interactivity an augmented reality sandbox is used to visualize the results and interaction with the model.

2. Motivation

Current Boussinesq models have promising results in modeling several coastal phenomena, however the heavy computational effort needed for Boussinesq-type equations hinders real-time simulations using them unless with parallel processing on dozens to hundreds of CPU cores. Such supercomputing facilities are neither easily accessible nor inexpensive. It should be also noted that current simulation models do not provide concurrent visualization of the results. Concurrent visualization becomes more significant and effective if interactivity is also provided, so the engineers and scientists can alter the water surface, bathymetry, simulation parameters, etc. on the go.

We aim to develop a software, called Celeris, which can be used for real-time wave simulation on an off the shelf laptop (Fig. 1). Celeris runs ~50 times faster than real-time for the modest coastal region with ~10^4 grid points.

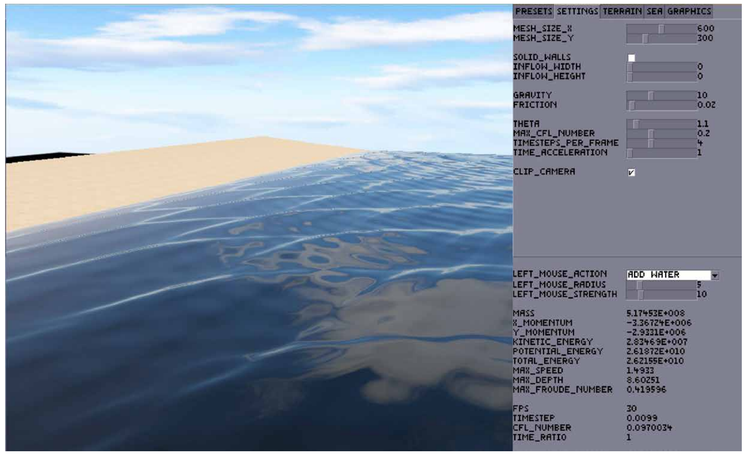

Figure 1- Snapshot of Celeris

3. Theory

The extended Boussinesq equations derived by Madsen and Sørensen (1992) for a slowly varying bathymetry read as:

$$h_t + P_x + Q_y = 0$$

$$P_t + \left(\frac{P^2}{h} + \frac{gh^2}{2}\right)_x + \left(\frac{PQ}{h}\right)_y + ghz_x + \psi_1 = 0$$

$$Q_t + \left(\frac{PQ}{h}\right)_x + \left(\frac{Q^2}{h} + \frac{gh^2}{2}\right)_y + ghz_y + \psi_2 = 0$$

$$\psi_1 = -\left(B + \frac{1}{3}\right)d^2\left(P_{xxt} + Q_{xyt}\right) - Bgd^3\left(\eta_{lxxx} + \eta_{lxyy}\right) - dd_x\left(\frac{1}{3}P_{xt} + \frac{1}{6}Q_{yt} + 2Bgd\eta_{lxx} + Bgd\eta_{lyy}\right) - dd_y\left(\frac{1}{6}Q_{xt} + Bgd\eta_{lxy}\right)$$

$$\psi_2 = -\left(B + \frac{1}{3}\right)d^2\left(P_{xyt} + Q_{yyt}\right) - Bgd^3\left(\eta_{lyyy} + \eta_{lxxy}\right) - dd_y\left(\frac{1}{3}Q_{yt} + \frac{1}{6}P_{xt} + 2Bgd\eta_{lyy} + Bgd\eta_{lxx}\right) - dd_x\left(\frac{1}{6}P_{yt} + Bgd\eta_{lxy}\right)$$

In the current work A hybrid finite volume – finite difference numerical scheme is used. The advective part of the equations are solved using finite-volume method, while dispersive and source terms are discretized using the finite difference formulation. This hybrid discretization enables the software to benefit from robustness of FVM, shock-capturing features, and flux limiters, while retaining the higher accuracy of the model. The model is also positivity preserving and therefore there is no need to keep track of dry/wet cells. Time integration is done with the third order Adams-Bashforth scheme as prediction step and the fourth-order Adams-Moulton algorithm as an optional correction step. It should be also noted that Boussinesq-type equations are not easily implemented on parallel computers, due to implicit methods necessary for their discretization.

4. GPU Implementation

In order to solve a computational problem with DirectX, the problem must be reformulated in terms of graphics primitives and data must be stored within textures. We use the R, G, B, A value of each texel to store flow parameters. Each step of numerical scheme is performed by passing a texture to a shader. A sample shader to apply boundary condition is shown in Fig. 2.

```
float4 WestBoundarySolid( VS_OUTPUT input ) : SV_TARGET
{
   const  float3 in_state_real = txState.Load(int3(4 - input.tex_idx.x,input.tex_idx.y,0)).rgb;
   return float4(in_state_real.r, -in_state_real.g, in_state_real.b, 0);
}
```

Figure 2- Sample shader code in HLSL

The most challenging part of the work is parallelization of solving the tridiagonal matrix systems within the numerical scheme. We accomplished it using the cyclic reduction (CR). CR consists of two phases, forward reduction and backward substitution. In the forward reduction phase the system is successively reduced to a smaller system with half the number of unknowns, until a system of 2 unknowns is reached which

5. GPU Implementation

can be solved trivially. In the backward substitution phase the other half of the unknowns are determined step by step using the previously solved values. In the present work CR is done by passing the texture of unknown variables to the reduction shader, iteratively, until two unknowns (equations) are remained. The remaining equations are solved and the results (known values) are passed to the substitution shader, iteratively, to find all variables. This process is illustrated in Fig. 3.

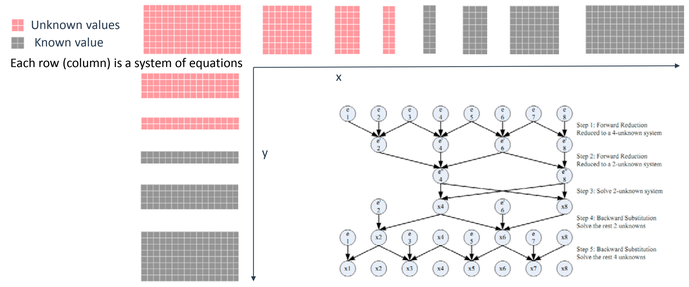

Figure 3- Cyclic reduction algorithm and its implementation on GPU.

6. Results and Validation

We validated our scheme with the experimental results of Synolakis (1987) for solitary wave runup on slope (Fig.4). The aim of current work is not to produce a tuned code on GPU with maximum performance. we compromise performance in favor of real-time visualization and inter-activity. As a preliminary experiment, we compared our GPU implementation of scheme with its single core CPU results on a machine with a CPU of Intel Xeon CPU @2.13GHz and a GPU of NVIDIA Quadro 600 (Fig. 5).

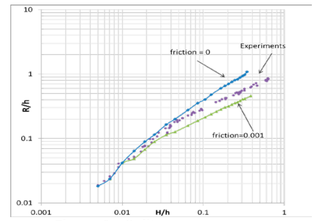

Figure 4 – Solitary wave runup

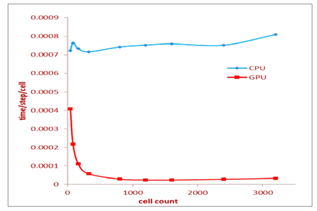

Figure 5 – GPU vs. CPU performance

7. Future work

We are about to take the next big step in engineering modeling by embedding an augmented reality sandbox (ARS) to the model. In ARS simulations results are cast on sand surface using a projector, and the sand surface play the role of the bathymetry (Fig.6). Our design is based on the ARS developed at UC Davis, but it is scalable and runs on Boussinesq equations. Scalability is delivered via threading and MPI system. Engineers can use as many sandboxes as needed together to model a larger region of a coast or a longer segment of a river. Engineers can modify the sand surface in order to see response in the water surface and its motion on the go. This moves the interactivity of the model to the next level. Currently we are trying to power the augmented reality sandbox using NVIDIA's Jetson TK1.

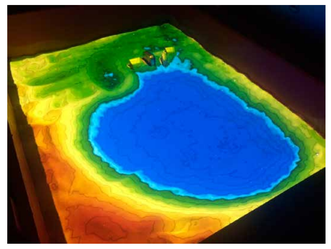

Figure 6 – Modeling a basin on sandbox