SIMULATION OF WATER FLOW OVER VARYING BOTTOM USING GRAPHICAL PROCESSING UNITS (GPUs)

M.N. Rud, V.R. Duseev
E-mail: rudmax13@gmail.com

Language supervisor: Butakova T.I.
Scientific supervisor: Malchukov A.N., Ph.D.
Tomsk polytechnic university, 634050, Russia, Tomsk, Lenin Avenue, 30

Introduction

One of the most complicated and challenging problems for rendering realistically looking terrain is water simulation. Below are some popular methods to perform this task:

1. Procedural water. It simulates the visual effects of water surface, but not the water physics. This approach is suitable for large unbounded surfaces simulation, such as oceans. The most realistic results were obtained by J. Tessendorf in 2004.
2. Smooth particle hydrodynamics. In this method water is represented by set of particles; the approach perfectly fits for simulating spray, splashing, and runnels. It is computationally expensive and not suitable for simulating rivers, lakes and oceans.
3. Height fields. The method represents water surface as a 2D-function $z = f(x, y)$. The advantage of this approach is transition from 3D-volume to 2D-surface, and, as a consequence, reduction of computational complexity. However, there is no possibility to simulate breaking waves, because at each point of surface there is only one height value.

The above methods have some advantages and disadvantages; it is necessary for a researcher to make the appropriate choice of the method according to the task, programming, mathematics skills and hardware resources.

Task

Our task is to develop a water model to be used on Interactive Sandbox, an installation for real-time terrain generating [4]. A good-looking and physically correct liquid model will add realism and functionality to the installation.

For our purpose, we need the water model, which satisfies the conditions below:

1. it provides realistic appearance of water;
2. water interacts with terrain and adapts to its changes in real time;
3. it is computationally as inexpensive as possible.
4. it allows simulating other liquids, for example, volcanic lava.

Shallow water equations

Shallow water equations (also called Saint Venant equations) are a set of hyperbolic partial differential equations describing the dynamics of a thin fluid surface in terms of its height and flow.

$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ -ghB_x - \kappa(h)u \\ -ghB_y - \kappa(h)v \end{bmatrix}$$

Figure 1. Shallow water equations

Here *h* is the water depth, *hu* is the discharge along the *x*-axis, *hv* is the discharge along the *y*-axis, *g* is the gravitational constant, and *B* is the bathymetry (see Figure 1).

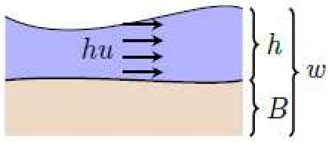

Figure 2. Variables for shallow water equations in 1D

The usage of shallow water equations has some limitations. As mentioned above we cannot simulate breaking waves and splashing particles, however, this method is perfect for modelling water domains whose surface area is much greater than its height. With the help of shallow water equations we can simulate not only small lakes, rivers, puddles, pools and ponds but also large surfaces of water.

Numerical scheme

We are interested in a simple, accurate, and robust numerical method for the Saint-Venant system. A good numerical method for the system should accurately capture both the steady states and their small perturbations (quasi-steady flows). From practical point of view, one of the most important steady-state solutions is a stationary one (lake at rest):

$$u=0,\ h+B=Const.$$

Methods having this property are called well balanced.

In addition, the method should handle dry (*h* = 0) or near dry zones (positivity preserving property). In these cases, due to numerical oscillations, *h* can become negative and all computations can simply break down.

Such a method which is both well-balanced and positivity preserving was suggested by A. Kurganov and G. Petrova in 2007 [2]. The system of differential equations is discretized on a regular Cartesian grid using the so-called central upwind scheme (Figure 3):

$$\frac{d}{dt}\overline{\mathbf{U}}_j(t) = -\frac{\mathbf{H}_{j+\frac{1}{2}}(t)-\mathbf{H}_{j-\frac{1}{2}}(t)}{\Delta x} + \overline{\mathbf{S}}_j(t),$$

Figure 3. Central upwind scheme of discretization

Here $\mathbf{U}_j(t)$ is a two-component vector of conserved quantities *w* and *hu* (three-component vector in case of 2D), where *w* is a sum of water height and bathymetry height at the point, and *hu* is the discharge along the *x*-axis. $\mathbf{S}_j(t)$ is an appropriate discretization of the cell averages of the source term, and $\mathbf{H}_{j+1/2}(t)$ is a central-upwind numerical flux along the *x*-axis.

After obtaining all necessary values, this system can be solved using a second order stability preserving Runge-Kutta method.

$$U_j^* = U_j^n + \Delta t(R(U_j^n));$$
$$U_j^{n+1} = \frac{U_j^n}{2} + \frac{1}{2}\cdot(U_j^* + \Delta t(R(U_j^*)));,$$

where $R(U_j)$ is the right part of equation in Figure 3.

Implementation using GPU

Unfortunately, implementation of Kurganov-Petrova scheme on CPU does not give us satisfactory results. Intel Core i5 processor can calculate only 50 vertices in each direction with acceptable frame rate (30 frames per second). This is not enough for two reasons: firstly, the surface is not smooth and does not look good enough, and, secondly, Kinect sensor that is used in Interactive sandbox produces the height map with resolution at least 320 x 240 points, so we need to obtain the appropriate resolution in water calculations.

GPUs have in recent years developed from being hardware accelerators of computer graphics into high-performance computational engines.

In our case we can use computations on GPU to implement Kurganov-Petrova scheme in parallel. We divide the entire algorithm into four stages. At each stage we perform calculations for each vertex independently from others. That is exactly how GPU can be used in non-graphic computations.

To implement the algorithm we use graphics API OpenGL and its feature called Frame buffer object. We use the first program to perform the first stage of algorithm in parallel for each vertex (the program performed on GPU is called *shader*). Then this data is used to fill the texture which can be used at the second stage. After all four stages proceed, we obtain the solution, which includes water height at this point, and use this value to render water surface.

Visualization

To visualize the liquid surface we use some well-known and not very computationally expensive techniques, which, nevertheless, give very good results.

1. To compute lighting we use standard ADS-model (which divides the light into three components – ambient, diffuse and specular);
2. We use skybox texture to simulate the effect of real environment around the scene.
3. For water to look realistically we use reflections of skybox and of terrain part which is above the water surface. To perform this we render the scene with terrain into the texture and then apply this texture to the water to obtain reflection.
4. As in real life, in our implementation water transparency depends on its height.

Blending liquids

Algorithm allows simulating two or more liquids on terrain simultaneously. Computations for them are performed in parallel, so there is no reduction of a frame rate. We cannot simulate physical interaction of fluids without particle system, but usage of specially computed color blending factor gives good visual results.

Results

The result of rendering can be seen in Figure 4.

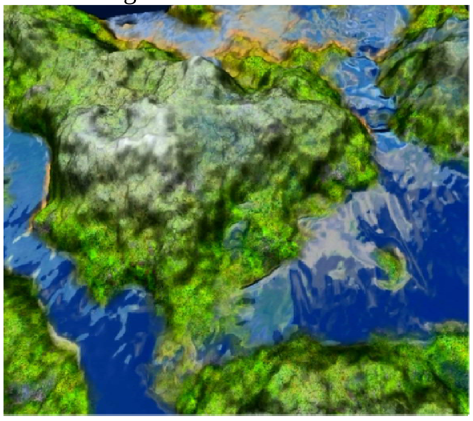

Figure 4. Final result of rendering

The water surface has resolution 320 x 240, obtained frame rate was 40 frames per second. The program was tested on the notebook with Intel Core i5 processor and NVIDIA GeForce 640M video card. The developed user interface allows user to add water in certain points with a mouse, change terrain, move lighting source, and tune water visualization to make it more transparent, and etc.

References

1. A. Kurganov and G. Petrova, “A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system,” Communications in Mathematical Sciences, vol. 5, pp. 133–160, 2007.
2. A. R. Brodtkorb, M. L. Sжtra, and M. Altinakar, “Efficient shallow water simulations on GPUs: Implementation, visualization, verification, and validation” 2010.
3. D. Shreiner, M. Woo, J. Neider, and T. Davis, OpenGL Programming Guide: The Official Guide to Learning OpenGL, 6th ed. Addison-Wesley, 2007.
4. Rud M., Duseev V., “Creating interactive visualization system”, TPU, 2013.