Realistic Simulation of Rivers

System Development Project (SEP)

Fabian Pache & Darko Zikic

pache@in.tum.de
zikic@in.tum.de

Prof. R. Westermann
Chair for Computer Graphics & Visualization
Technical University Munich

Supervisor: Peter Kipfer

Start of project: 01.03.2003
End of project: 01.09.2003

Contents

1 Introduction

The task of our SEP (German short for System Development Project) was to develop and implement a realistic simulation of rivers on arbitrary terrains on a normal desktop PC[1]. Since nowadays it isn't yet possible to achieve speed and scientific accuracy at the same time we had to find an inexact but still believable, "good looking" and fast model to accomplish our task. After considering several possibilities we decided to use a particle system approach and test several physical models.

2 Discussion of possible Approaches

After basic research on the topic we had three possible alternatives for our simulation model.

- The "classical" numerical solution of the Navier-Stokes fluid equations.
- Smoothed particles hydrodynamics (SPH) – an approach using a particle system.
- Another model also using a particle system yet simpler than SPH (in the following referred to as Simple Model).

Solving Navier-Stokes equations was not appropriate for our problem. Although very exact its computation in three dimensions takes a lot of time and an efficient implementation for arbitrary terrains is very complicated. ([GDN95], [Sch03])

The Smoothed Particles Hydrodynamics seemed a good approach. It doesn't require a grid structure as the Navier-Stokes equations but acts on a particle system in which a single particle stands for a certain volume of the fluid[2]. It is sufficiently exact and faster than Navier-Stokes. [SAC+99], [DG], [Roy95], [Mon])

The last alternative was actually our very first idea. We use a particle system where particles are supposed to be something like water balloons. It should provide a simple model to evaluate the basic functionality.[3]

For speed reasons we decided to follow the Langrangian approaches (SPH & Simple Model) and build a framework in which basically any particle system based model can be used.

3 Basic Concept

The framework has the following structure:

- for every particle: find the particles colliding with it *(collision detection)*
- for every particle: apply the underlying physical model to the particle and its neighbours *(collision treatment)*

[1] this means a 2GHz, 512MB RAM, GeForce FX machine

[2] Methods following the physical values at the particles are referred to as Langrangian while methods evaluating values at grid points like the Navier-Stokes method are called Eulerian

[3] The vision behind the idea was that if you have really many water balloons and throw them all down the hill and go far away enough they will seem like a river.

- for every particle: update the position of the particle according to the set acceleration and speed values *(integration step)*
- for every particle: check for collision with the terrain and act appropriately *(terrain collision)*

Hence the basic parts of the project are:

1. Simulation
    - Collision detection
    - Collision treatment (physical model)
    - Terrain Collision
2. Visualization
    - finding the surface for the particles

Now we will discuss those points in more detail.

4 Simulation

4.1 Collision Detection

An efficient collision detection is crucial for the simulation to run fast. The naive approach that tests for collision detection between one particle and all others has costs of $O(n^2)$ and is thus not acceptable for the large number of particles needed.

Our approach is the following. We split up the space in cubes with edge length of 2r (r being the maximum radius of a particle). This way all potential neighbours of a particle with a center inside one cube must be in the surrounding 26 cubes plus the cube containing the particle itself.[4] Every cube has a list of particles whose centers lie within so finding the neighbours of a particle is trivial. Because we only have to check the particles of the surrounding cubes for collision detection we obtain a collision detection algorithm with linear cost.

4.2 Collision Treatment

When a collision occurs that means that the two particles are influencing each other in some way. The way they are doing this depends on the physical model used. The physical model sets the particle's values for acceleration and speed and then the integration step is done for all particles. We implemented and run tests with two different models, SPH and the Simple Model. Comparing the two models the Simple Model is surely much more inaccurate but has the advantage that it is quite simple and thus it is much easier to reach believable results with it. Besides it is possible to achieve larger time steps and it is faster since it has simpler and less calculations per step.[5]

As described above the Simple Model is a very simple model assuming that the single particles are something like water balloons. It has no real physical justification. Its only aim is to produce simple, fast and realistic looking results.

[4] If particles of different size are used there is certain overhead for the smaller particles.

[5] The Simple Model has less calculations because it is possible to reduce the number of collisions per particle while the particles in the SPH model basically always interact with their neighbours.

To achieve this we had several requirements for the model. Mainly they were the behaviour due to the impulse when particles are colliding, the gravitational force when they are in contact and over each other and pressure forces when overlapping. Another requirement was that the volume of the fluid is conserved as good as possible and of course that all these conditions together result in a altogether believable behaviour.

4.2.1 Speed and direction change after collision

When a collision between two particles occurs momentum is transfered from one to the other. For determining speed and direction of two particles after a collision we use a modified version of the (imperfect) inelastic collision model[Tip99]. It is a combination of an elastic and perfect inelastic collision.

4.2.2 Influence due to the gravitational force

If one of the particles is above the other the upper particle additionally exerts a gravitational force on the lower particle.

4.2.3 Pressure forces

If the collision treatment doesn't suffice and the particles are still overlapping then this is interpreted as high pressure on that point and the upper particle is accelerated upwards.

4.2.4 Volume conservation

Incompressibility and thus volume conservation is an important characteristic of water. We tried to reach that nearly by adjusting the interaction between particles so that they stay as close to each other as possible without overlapping.

Not only for physical plausibility but also for performance reasons it is important that particles do not overlap and that every collision is solved in one step – or at least in as few steps as possible. If attained, this reduces the number of collisions per step and thus speeds up the simulation and/or enables us to use a larger number of particles. Unfortunately many attempts and strategies to assure that particles are absolutely not overlapping and that every collision is treated in one step were not successful because of their side effects. The ratio of collisions per step to number of particles depends heavily on the terrain used. For deeper lakes it is higher than for shallow fast streams. In our simulation we achieved to keep this factor less than 3.

4.3 Terrain Collision

Terrain collision is just another thing that takes quite a lot of time if you want to do it right[Ebe00]. Since we are not willing to sacrifice too much time for this operation we don't do an exact collision. For an exact terrain collision we would have to calculate the exact point where the particle (a ball or at least a point, if we consider the center of the particle) collides with the terrain on its path. To do it properly a "triangle vs. moving sphere" collision would have to calculated for the set of triangles the particle passes over for a given time step. This obviously leads to a very time consuming search and evaluation operation.

Instead of implementing the exact algorithm we use an iteration that approximates the collision point accurately enough. First we assume that the particle was in a valid

position, that is above ground, before the last integration step. As long the particle's position is illegal (particle under ground) or not accurate enough (particle above ground) we iterate along the path the particle took to its current position.

In little bit more detail we do the following. We just look whether the center of the particle is under the terrain after it moved. If it isn't then everything is just the way it should be and no terrain collision occurred. Otherwise the particle is stuck in the terrain and we go back half the way the particle moved. Then again, there are two possibilities. If the particle is under the terrain again we go quarter the way back. If it's above the terrain we now move quarter the way forward. We repeat this by taking always half the way as in last step until our desired accuracy is reached. The outcome of the iteration is a particle that again has a valid position and therefore satisfies out assumption of a legal position for the next step of the integration. Particles for which no valid position can be found are removed from the simulation. We consider these particles soaked up by the underlying terrain.

5 Visualization

The visualization of the fluid is a crucial point. Previous implementations using implicit functions for visualization experienced that more time can be needed for the visualization than for the simulation itself.[Roy95] Again we have to make a compromise and accept some backdraws in order to achieve acceptable speed.

The idea we had is to lay a sort of carpet over the particles. The particles push the carpet up and the carpet falls down if there is nothing underneath to support it.

Since the carpet can easily be rendered as a triangle/quad mesh and the pushing up by carpets is as simple this certainly is a very efficient approach compared to other methods like solving an implicit function over particles or ray tracing.

But there are also advantages of this method that go beyond mere speed. It proved to be very suitable for visualizing rivers because the implicit function method always gives an impression of a too viscous fluid. Furthermore it allows for further effects like surface waves or visualize additional information like the primary movement direction and speed of the underlying particles[6]. In fact the mere pushing up of the carpet by the drops already produces a wave like effect and provides an impression of motion. And last but not least it is possible to use the altered volume to create the illusion of a much denser particle system than really present. Gaps between particles that are greater than the sum of the radii of the particles are smoothed over as long as the particles are moving coherently.

However here are also backdraws of the method. The two major are that the carpet method can only be used to visualize the upper surface of a fluid and that the method is not volume conserving.

The first backdraw is not such a problem for simulating rivers.[7] The method reaches its limits only with phenomena like waterfalls or fountains but remains however applicable in non-extreme cases.

The problem of volume non-conservation is more important for simulation of rivers since it is noticable during events like a filling of a lake. A possible solution would be the make the change of the carpet by a drop dependent on the pressure value of the particle.

Since our simulation doesn't run in real time we provide possibilities to take pictures

[6] similar to the Line Integral Convolution method

[7] What's the last time you saw the bottom surface of a river?

at a rate of 25 frames per second of the simulation time which than can be converted to a movie showing the "real" flow. There is an option to export either BMP files or PovRay[8] scene files.

[8]PovRay - Persistence of Vision Ray Tracer, www.povray.org

6 Appendix - User Manual

There are a lot of commands and options that can be set at runtime. None of the settings you make are saved for your next session. You will welcome this as a feature as soon as you have fed the random terrain generator with impossible values and can not find the original values. Maybe even sooner. At any time you can quit the program by pressing `Q` in addition to any commands your operation systems offers to shut down applications politely.

6.1 Command Line Options

There are no options that can be passed to the fluid simulation from the command line. However the parameters are passed to the GLUT extension. Please refer to the GLUT manual for valid options.

6.2 Controlling the Viewpoint

There are two modes of control for the viewpoint. The Examination mode, or camera, allows rotation around an arbitrary point while the Personal mode allows for more direct control. The modes can be toggled using the `F4` key. The following table shows keys and their effects for each of the camera modes.

key	Examination	Personal
`up/down`	raise/lower	look up/down
`left/right`	rotate right/left	look left/right
`+/-`	zoom in/out	move forward/back

Holding the left mouse button and dragging the mouse inside the window has the same effect as the `arrow keys`. Holding the right mouse button instead works like the `+/-` keys. Using the mouse much finer commands can be issued. The following commands can not be issued using the mouse

key	Examination	Personal
`Home/End`	move forward/back	raiser/lower
`Insert/Page up`	move left/right	raiser/lower
`S`	reset viewpoint position	
`V`	reset viewpoint orientation	

6.3 Fluid Control

By pressing `F3` some mayor aspects of the fluid, especially its source, can be set. While in this mode a small blue fountain icon is displayed in the lower left corner of the viewport to show that the normal camera controls now apply to the fluid. Leave this mode by pressing `F3` again (surprise)

key	Fluid Control
up/down	move source toward/away from viewpoint
left/right	move source left/right from viewpoint
+/-	increase number of emitted particles per second
Home/End	increase/decrease emit speed
Insert/Page up	decrease/increase emmitter
,/.	rotate emitter
S	reset fluid
V	reset source

While the commands above are only available while in the Fluid Control mode the following are valid regardless of the selected mode

key	Fluid function
a/y	decrease/increase the simulation stepsize
l/k	decrease/increase the size of the particles
I	Restart the simulation. Resets all counters and clocks as well as the fluid

6.4 Terrain Creation

There is an almost infinite number of terrains available for simulations. Taking the random generated terrains into account the number is really infinite. Small wonder terrain creation is a bit complex but thats a small price for such a powerful tool. First some settings that apply to all terrains:

key	Function
F1	Display a short summary of commands and terrain generating options
F2	Generate mesh using the parameters currently set
F8	Switch through the available Terrain resolutions
F10	Switch through the various datasources
F11	Switch through the various blenders

A note on datasources and blenders: A datasource is either a flat plane, an image or a noise generator. A blender can be any function that takes a position and the datasource value and computes the final height value of the terrain at that position. The datasource "Noisefield" is somewhat special in that it takes further parameters. The "noisefield" is a Perlin-Noise tool that uses the following parameters:

key	Function
F5	Reinitialize the noise generator
F6	Increase the lower bound of the frequencies used by the noise generator
F7	Increase the frequency range used by the noise generator
F9	Toggle the layering of frequencies used by the noise generator

Since noise generation is not a topic of this SEP, those terms are not explained here. If the terms of a Perlin-Noise generator are unfamiliar to you I suggest you track down some tutorials on Ken Perlins work or start right at his page.

6.5 Creating an Animation

There are two methods of creating an animation. Either export every single image of the simulation as a bitmap or create a sequence of pov-ray files that can be used to raytrace

the flow. The two methods do not produce the same images as the pov-ray export uses raytracing and realistic water properties to create the fluid while our internal drawing uses our own carpet without refraction, reflection and the like.

key	Function
B	Export the heightfield for pov-ray
E	Switch through the 5 available timermodes
M	Start/Stop recording a sequence of bitmaps.
N	Start/Stop exporting the fluid for povray.

6.6 Visuals

The following commands permit the user to change the look of the simulation.

key	Function
C	Toggle the sky. Looks good but doesn't affect the simulation.
F	Switch through the 3 available fog modes
G	Switch through the various fluid display methods
H	Display the normals of the terrain.
T	Change texture. Either a canyonish brown or something blueish
W	Toggle though the wireframe modes and solid rendering

References

[DG] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies.

[Ebe00] D. Eberly. *3D Game Engine Design*. Morgan Kaufmann, 2000.

[GDN95] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerische Simulation in der Strömungsmechanik*. Vieweg, 1995.

[Mon] J.J. Monaghan. *Smoothed Particle Hydrodynamics*. Annual Reviews Inc.

[Roy95] T.M. Roy. Physically based fluid modelling using smoothed particle hydrodynamics. 1995.

[SAC+99] Dan Stora, Pierre-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. Animating lava flows. In *Graphics Interface*, pages 203–210, 1999.

[Sch03] Thomas Schiwietz. Echtzeitfähige simulation von wasser auf grafikhardware. 2003.

[Tip99] Paul A. Tipler. *Physics*. Freeman Worth, 1999.