

Simulation, Modelling and Rendering of Incompressible Fluids in Real Time

Thomas Klein, Mike Eissele, Daniel Weiskopf, Thomas Ertl

Universität Stuttgart

Institut für Visualisierung und Interaktive Systeme (VIS)

Universitätsstraße 38, 70569 Stuttgart, Germany

Email: {klein,eissele,weiskopf,ertl}@informatik.uni-stuttgart.de

Abstract

In this paper we present techniques for the real-time simulation and rendering of liquids. Appropriate approximations to a full 3D simulation are applied to reduce the numerical complexity. Fluid flow is described by the 2D Navier-Stokes equations; wave effects are represented according to the wave equation for shallow water waves on a height-field. A novel mechanism for coupling flow and wave behaviour is introduced to efficiently handle transport along a fluid flow and accurate wave propagation on the fluid surface. An additional noise-based animation of detailed fluid structures further improves the realistic appearance. Rendering makes use of fragment operations on consumer-level GPUs to allow for a physically guided representation of reflection, refraction and Fresnel blending of light.

1 Introduction

Water is present in our everyday lives. Therefore the realism of any virtual environment is greatly enhanced by including water phenomena, and the animation and rendering of liquids is an important and challenging topic in computer graphics. In recent years much progress has been made with the physically based simulation of water effects. Most techniques are based on direct simulation methods to solve the full 3D Navier-Stokes equations for incompressible fluids, e.g. [6]. Recent publications concerning fluids mostly make use of a semi-Lagrangian treatment introduced to computer graphics by Stam [18]. This method avoids numerical oscillations in convection-dominated flows and allows large time steps. State-of-the-art animation and rendering of water using 3D Navier-Stokes is

presented by Enright et al. [4]. The major disadvantage of these accurate simulation methods are the required computation and rendering times.

In this work we do not focus on a physically correct simulation but rather on real-time animation and rendering of realistic looking water effects. There is a large, commercially interesting market for such applications, reaching from the game industry to visual simulation applications and interactive controlling and modelling of liquids in animation tools. Typical performance should be some 5–60 frames per second depending on the type of application, i.e., between two to three orders of magnitude faster than a full-bodied simulation. Even with the increasing performance of PC hardware in the years to come, a real-time application of a complete 3D liquid simulation seems to be impossible on PCs in the foreseeable future. Therefore we propose an approach that allows for a restricted, yet important class of water phenomena. The main idea is to reduce a full 3D Navier-Stokes simulation to an appropriate 2D approximation. A two-dimensional computation is less expensive than a three-dimensional and, in particular, scales much better with increasing numerical grid resolution.

In this context one class of previous and related animation approaches focuses on a reduced model representation of fluid surfaces by means of parametric modelling [7, 14, 20]. Another approach [13], guided by oceanographic observations, is based upon the filtering of a noise field by a wave spectrum using the fast Fourier transform. In recent work [10] an adaptive scheme for the real-time animation and display of ocean waves is proposed. Common to all above methods is that they contain no concept of the fluid flow or mass transport, but only a pure description of the fluids surface behaviour.

Another class of techniques is based on 2D approximations of the full 3D Navier-Stokes equations. The approach presented here follows this line of reasoning and is closely related to the following two papers. Kass and Miller [12] obtain a height-field representation of the fluid surface by means of a linearised form of the 2D shallow water wave equations. Chen and Lobo [2] use the 2D incompressible Navier-Stokes equations to compute a 2D flow and directly drive a height-field by the pressure scalar field obtained from the flow simulation.

An important contribution of this paper is to introduce a coupling between flow and wave behaviour. In our system, flow properties can induce wave effects in order to consider, e.g., an inflow into a lake that produces both streaming and wave effects. Using a height-field restricts the possible classes of phenomena that can be achieved, e.g., splashing or pouring cannot be reproduced. However, visually important effects of wave propagation are taken into account and moreover, transport along the flow of the liquid is considered, by means of the flow simulation based upon the 2D Navier-Stokes equations. A typical application is the simulation of the flow and waves for a river or a lake.

Additional noise-based animation techniques are used to further improve the appearance of the fluid surface. Noise-based animation has been found useful for approximating a wide range of natural phenomena. Stam and Fiume [19] use an FFT method to add a small scale component to turbulent wind fields. Also, Schneider and Westermann [16] generate a water surface by completely restricting themselves to noise in the form of fractional Brownian motion. We use this approach to add some additional detail structure to our simulated fluid surface.

Besides the animation of fluids the visual representation of their surfaces is extremely important for the realism of a graphical application. There are many rendering algorithms that can properly simulate complex light transport, but due to their computational complexity most of these algorithms are not suitable for real-time applications. We propose a hardware-accelerated approach to allow for physics-based reflection and refraction effects, which is faster than a related approach [11]. The computations are performed with per-pixel accuracy and exploiting the capabilities of low-cost graphics hardware.

2 Numerical Description of Fluid Flow

The dynamic behaviour of a viscous fluid, like water, is completely described by the so-called Navier-Stokes equations. For incompressible Newtonian fluids they state that, in every point, the following conditions must be fulfilled,

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla) \vec{v} = \frac{1}{Re} \nabla^2 \vec{v} - \nabla p + \vec{f} \quad , \quad (1a)$$

$$\nabla \cdot \vec{v} = 0 \quad , \quad (1b)$$

where \vec{v} is the fluid velocity, p the pressure, \vec{f} the external body forces and Re the so-called Reynolds number, which accounts for the overall behaviour of the flow. For a detailed derivation and description of the Navier-Stokes equations we refer to textbooks on fluid dynamics, such as [1, 8].

This system of partial differential equations has to be solved for the velocity \vec{v} and pressure p in every point of the computational domain. In general there is no analytical solution, so numerical methods have to be employed. In this work a finite-differences approach based on the marker-and-cell (MAC) method by Harlow and Welch [9] is followed. Spatial derivatives are discretised using central differences on a staggered grid.

But employing central differences for discretising all spatial derivatives leads to problems, like numerical oscillations for convection-dominated flows [1, 8]. Therefore we propose a combination of central differences and a donor-cell scheme to discretise the convective term $(\vec{v} \cdot \nabla) \vec{v}$. The donor-cell scheme is well-known in the CFD community, cf. [8], and has been used, in a slightly different form, for example in [21].

The qualitative behaviour of the donor-cell scheme can be compared to the semi-Lagrangian methodology [18]. Combining the donor-cell scheme and central differences has the nice property, that it provides a user-adjustable factor that can be used to find a compromise between the numerical damping due to first order interpolation and the second order accuracy of central differences.

The development in time due to Eqns. (1) is treated by a standard projection method [3, 8]. According to that, the numerical algorithm for solving Eqns. (1) can be split into three major steps. First, a temporary velocity field \vec{v}_{tmp}^{t+1} is computed by applying an explicit Euler method, while neglecting the

coupling to the pressure. Second, a Poisson equation for the pressure field is derived and solved by treating the pressure derivatives implicitly in time and enforcing conservation of mass by means of Eqn. (1b). Third, the new velocity field \vec{v}^{t+1} is computed by updating $\vec{v}_{\text{imp}}^{t+1}$ using the new pressure values obtained in the previous step.

For a visual real-time simulation, a constant computation time per frame should be reached to facilitate a constant application flow. The computation time for steps one and three is of $O(n)$, where n equals the number of grid cells. However, computation time for the second step is rather problematic. A numerical approach to the Poisson equation leads to a sparse system of linear equations which could be solved by a number of different methods. With respect to performance, iteration methods are better suited than direct methods for such sparse systems. A popular method is SOR iteration (successive over-relaxation) [22], for which the number of iteration cycles depends on the error level and is not constant. In our approach, on the other hand, only a fixed number of steps of the iterative algorithm is computed regardless of the achieved convergence. Although this does not lead to an accurate solution, because the continuity condition is not completely satisfied, the achieved accuracy is still sufficient for a qualitatively correct fluid behaviour in a visual simulation.

Since the SOR iteration has a preferred direction of iterating through the solution vector, artifacts can occur if the iteration is terminated before convergence is reached. These artifacts can be reduced by alternating the direction of updates in every iteration step, which leads to the symmetric SOR (SSOR) method [22]. From experience, six to eight iteration steps are sufficient to achieve results with sufficient accuracy for typical visual simulations of fluids. The fixed number of iteration steps guarantees a constant execution time.

Another issue is numerical stability. Because we use a semi-explicit time integration scheme, our method exhibits the inherent instability associated with explicit methods. The maximum time step is restricted by various factors, including the Reynolds number, the cell size, the donor-cell damping factor and last but not least the applied boundary conditions. We do not present a thorough stability analysis of the solution scheme; the interested reader may refer to the CFD literature, for example [9, 1, 8] for

stability conditions that have to be satisfied. But it should be noted that due to our specific simplifications in the pressure correction step these conditions may not be very reliable. Generally it can be said that small time steps, lower Reynolds numbers, small inflow velocities, high donor-cell damping factors and good-natured boundary conditions lead to stable solutions.

3 Shallow Water Waves

For simulating the fluid surface we use a method proposed by Kass and Miller [12] to compute the wave propagation on a height-field. This method is based on the so-called shallow water equations, a simplified form of the previously described Navier-Stokes equations. Introducing several further assumptions these equations can be reduced to a single wave equation:

$$\frac{\partial^2 h}{\partial t^2} = gd \left(\frac{\partial^2 h}{\partial x^2} + \frac{\partial^2 h}{\partial y^2} \right) , \quad (2)$$

where h is the deviation of the surface from the fluid at rest, d the water depth and g the gravitational constant.

This wave equation can be efficiently solved using the ADI (alternating-direction implicit) method [12]. The advantage of the ADI method over a simple explicit method is its unconditional stability which results from an implicit discretisation. Moreover, computational costs are linear to the number of grid cells. Therefore a constant simulation time can be achieved for real-time applications. We introduce additional damping via a Lax-Wendroff dissipative interface as described in [1] to account for the viscosity of the fluid.

4 Coupling Flow and Wave Behaviour

So far two distinct methods that describe some partial properties of the fluid behaviour have been presented: Wave propagation on a height-field and 2D flow. The fundamental problem one encounters at this point is how to couple the pure 2D flow with the height-field in 3D. Considering real fluid motion one observes that fluid flow and surface waves are two, although not independent, yet loosely coupled fluid properties. Waves have virtually no effect on the flow itself because they exhibit no mass transportation. The underlying fluid flow instead

is—apart from other sources like wind or bodies falling into or moving through the fluid—the main reason for the existence of surface waves. So the surface waves can be considered as a minor effect compared to the fluid flow and therefore one can focus on a mapping from 2D flow to wave effects on the height-field. In [2] this problem is solved by linearly mapping pressure to height values. This approach has the drawback of a direct flow-to-wave relationship that does not allow for effects like superposed waves originating from other locations of the flow or outside sources (objects falling into the liquid, waves reflected at boundaries, etc.). Instead we propose an indirect coupling between the 2D flow and the height-field. The solution of the flow simulation is utilised to generate initial disturbances in the height-field that are propagated by means of the wave equation.

We use the local divergence of the temporary velocity field $\tilde{v}_{\text{tmp}}^{l+1}$ generated by the Navier-Stokes solver to disturb the height-field. This follows a very simple heuristic: divergence indicates a change in mass, i.e., mass has been moved to or from neighbouring cells. This mass divergence must result in a change of the surface height. If there is a mass surplus (negative divergence) in the cell, the fluid surface has to rise; vice versa if the divergence is positive, the surface height must fall. The disturbed height-field before the wave propagation step is then given by

$$\tilde{h}^{l+1} = h^l + f_h(\nabla \tilde{v}_{\text{tmp}}^{l+1}) \quad ,$$

where the function f_h defines how the current height-field h^l is affected by additional disturbance. Defining $f_h(\delta) = s\delta/(1 + (h^l)^2)$ has proved to be useful because this mapping takes into account that the greater the deviation between the actual fluid surface and the fluid surface at rest is, the smaller is the effect of any additional disturbance. The strength of the perturbation can be controlled via the scaling factor s . Other choices for the mapping function f_h are possible as well. In particular, the direction of the deviation compared to the direction of the newly introduced disturbance can be taken into account in addition to the magnitude of deviation. For example,

$$f_h(\delta) = \begin{cases} \frac{s\delta}{1+(h^l)^2} & \text{for } \text{sign}(\delta) = \text{sign}(h^l) \\ \frac{s\delta}{1+(h^l)^2} & \text{for } \text{sign}(\delta) \neq \text{sign}(h^l) \end{cases}$$

takes into account that moving further away from the equilibrium state is considerably harder than moving towards the fluid height at rest.

The combination of a 2D Navier-Stokes based flow simulation and a wave equation driven height-field allows us to benefit from the advantages of both methods. Using the wave equation enables one to take advantage of a wide range of wave phenomena. This includes, e.g., wave refraction due to ground unevennesses, reflection of waves at obstacles or superposition of waves originating from wind force or fluid-user interactions. In contrast, the flow simulation can be employed for material transport, e.g., by tracing massless particles—which can be used for simulating leaves flowing on water (cf. Fig. 4). Another application is the interaction between the fluid and rigid bodies flowing within it.

5 Controlling the Flow

The behaviour of the fluid flow is mainly controlled by setting boundary conditions. Three different types of boundary conditions are supported: Each cell on the computational grid can be set to be an inflow, an outflow or an obstacle cell. These boundary conditions are applied during each simulation cycle, i.e., values for velocity and pressure are set according to the respective conditions.

Setting boundary conditions can be done in two different ways. First, the simulation system itself can set obstacle cells automatically depending on the geometry of the simulation environment. The data describing the geometrical properties of the simulation environment is specified in the form of a 3D scene file (in the current implementation a simplified variant of the Autodesk 3DS file format). It must contain the geometry of the scene surrounding the fluid and one or more planar meshes that define the geometries of the fluid surfaces. When the scene is loaded, each fluid surface gives rise to an appropriate rectangular grid whose cells are classified either to be fluid or obstacle cells. Fig. 1 illustrates how the grid is generated from a triangulated fluid surface. Here, the cells completely covered by the original mesh are regarded as fluid cells. The other possibility is to set the cell type interactively before or even during the simulation run. For this purpose our system provides a cell-based editor that can be used to manipulate the types of the grid cells and to set their properties, e.g., time-dependent inflow ve-

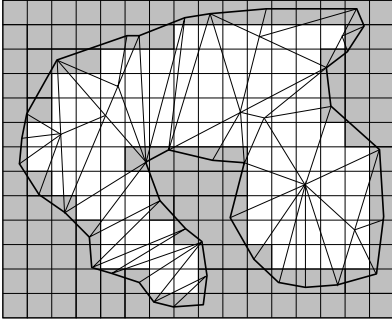


Figure 1: Automatic boundary condition setup.

locity. Moreover the editor allows the user to add sources for massless particles or rigid bodies.

6 Rendering

High-quality rendering of the fluid surface is crucial for a realistic looking simulation. Important optical phenomena related to the interaction between light and a fluid surface have to be taken into account by a physical model.

We present a solution that allows for effects of reflection, refraction and Fresnel blending. All computations are based on the limited per-fragment operations of today’s pixel shader 1.0 compliant consumer graphics hardware such as the GeForce3. The geometry of the fluid surface is built from the height-field over the uniform grid that is calculated in the simulation part.

The calculation of the aforementioned effects take place in several different coordinate systems. Object space and world space are known from the standard rendering pipeline. We need two additional spaces: surface-local space [5] and refraction space. The surface-local space, also known as tangent space or texture space, is defined for every point on the surface such that the point lies in the origin and the normal vector points along the positive z axis. The x and y axes lie in the tangent plane and are orthogonal to each other. The setup of the refraction space is illustrated in Fig. 2. The y axis is oriented in a way to make the incoming line of sight \vec{V} point along the negative y axis.

Fragment operations allow to calculate the reflection vector for a given viewing vector and surface normal. Since we apply the normals per fragment

in surface-local space, we have to transform them to world space. The so calculated reflection vector points into a cube texture map to provide the reflection colour according to the environment mapping approach. Unfortunately, refraction cannot be im-

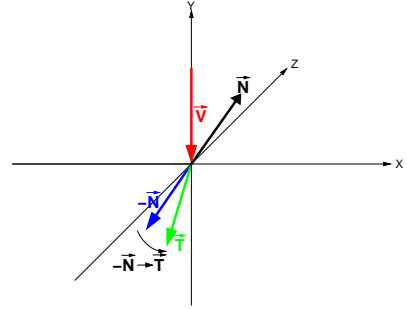


Figure 2: Refraction space.

plemented in a similar, straightforward way. The aforementioned class of graphics hardware imposes rather sharp restrictions onto the number and types of possible fragment operations. Due to these limitations, the refraction vector cannot be computed and used for a lookup in an environment map. To overcome this problem we propose an approximation based on the refraction space as illustrated in Fig. 2.

The negative of the normal vector, $-\vec{N}$, represents the transmission vector for a refraction index of infinity; the vector \vec{V} represents the transmission vector for a refraction index $n = 1$ (no refraction). The transmission vector for a refraction index $n \in [1, \infty)$ has to lie between $-\vec{N}$ and \vec{V} . This observation leads to the following qualitatively correct approximation of the transmission vector: the negative normal $-\vec{N}$ is scaled down in its x and z components to yield the approximated transmission vector. This non-uniform scale forces the negative normal to converge to the transmission vector. The amount of the scale corresponds to the refraction index.

The approximated transmission vector is used for lookup in the cubic environment map to collect the refraction colour. The advantage of this approach is the missing computation of complicated numerical terms that are replaced by a linear transformation via a combination of several matrices $\mathbf{M} = (\mathbf{R})(\mathbf{T})(\mathbf{R}^t)(\mathbf{S})$. The matrix \mathbf{S} transforms the normals from surface-local space to world space. The

matrix \mathbf{R} transforms vectors from refraction space to world space. A non-uniform scale of the normal in refraction space is performed through the matrix \mathbf{T} . The final matrix \mathbf{M} transforms the surface-local normal vectors to world-space transmission vectors.

A fluid surface that is only represented through the height-field from a simulation often misses to display fine details on the surface. A typical resolution of the computational grid is some 128^2 cells for a real-time simulation on a current PC. This resolution is roughly one to two orders of magnitude smaller than a typical screen resolution. However, fine surface details contribute a huge amount of realism to the appearance of the fluid. A realistic and efficient possibility to add detail to the surface makes use of a noise-based approach. The per-fragment normals of the fluid surface are perturbed, in order to achieve a realistic appearance. We choose a turbulence function consisting of different frequencies of Perlin noise [15]. Blending different scales of the same original noise field yields the different frequencies. This method is well suited for a real-time implementation because all necessary computational steps are supported by graphics hardware.

So far only a static perturbation of the fluid height-field is possible which would look totally unrealistic in an animation. To overcome this problem the surface detail is animated with a time-dependent spatial offset for the different frequency patterns. The combination of these patterns is computed for each frame with changing offset values. The result is an animated random surface detail where the viewer is no longer able to recognise a single random pattern.

Moreover, a computation of the overall noise texture for each frame allows us to include time-dependent local distortions that only show up in a spatially limited region. For example, a small gust of wind can have a local effect on the appearance of the fluid surface because wave ripples are introduced by wind dragging. These local effects massively contribute to a realistic appearance of the whole fluid. The local features very often concern high frequency noise that is modulated upon the random surface detail, as illustrated in Fig. 6. A static shape of the local distortions with sharp edges would lead to an unrealistic impression. To circumvent this behaviour the intensity of the effect is faded out at the border and the shape is animated

using a weighting factor represented through a 2D texture. The position, size, motion and shape of local effects can easily be managed by changing this texture. An efficient approach to animate the shape of the effect is to blend several static shapes with different blending factors. Therefore the surface appearance can be controlled by outside effects and react to conditions of the surrounding environment.

Finally, a lighting term is evaluated per fragment to modulate the refraction colour. This allows us to simulate the lighting of the ground through the fluid. Afterwards the blending of the reflection and refraction terms is achieved via a approximated Fresnel term $F = (1 - \cos \theta_i)^4$, since an evaluation of the correct Fresnel term using only fragment operations is not possible on the targeted graphics hardware. The value θ_i describes the angle between the surface normal and the viewing direction. In most cases, the viewer cannot distinguish the true solution of the Fresnel term from this rough approximation, whereas the calculation of this simple approximation uses only a few fragment operations.

A single cubic environment map is used for the reflection and refraction computations. This environment map has to be updated whenever the viewer's position is changed. Otherwise spatial distortions may become apparent if the position of the viewer is not equal to the position from where the environment map was created. Note that this dynamic environment map is generated once per frame and can be re-used for other rendering effects of a 3D engine.

7 Implementation and Results

A key element of the implementation is the partitioning into simulation and rendering parts. The implementation of the simulation part is based on C++ and makes use of the CPU only. Rendering is based on a GeForce3 GPU using OpenGL with several extensions. This separation leads to a parallelisation where the CPU calculates the simulation while the GPU renders the previous step.

The rendering part is divided into four steps. The first step for every frame is to update the dynamic environment map. For fast updates of dynamic textures any data transfer between the GPU and the CPU should be avoided. Since the functionality to directly render into textures is only available for some operating systems, we use a `PBuffer` to ren-

der into an offscreen area.

In the second step the surface detail texture is updated. Two stages of the static noise texture are blended with different scaling and offset values. With two additional stages a spatially limited change of the detail texture is introduced. The first of these two texture stages is setup to blend three static shapes of a local feature which are encoded in the colour channels of a texture. The following stage uses the blended shape to mask the high frequency noise texture for the local effect. The surface detail now contains perturbations of the normals for each fragment of the fluid surface. This normal texture is used in the following rendering passes for reflections and refraction.

The reflection and refraction rendering passes use vertex programmes to efficiently calculate the normals of the height-field. OpenGLs vertex pointers provide a fast access to the scalar height-field in order to evaluate central differences at the vertices. The x , y and z components in the vertex pointers hold three, shifted versions of the height-field, such that the y components of the first and third pointer and the x and z components of the second represent the height values for the direct neighbours of the current vertex. The y component of the second pointer provides the height of the vertex itself. Thus, the vertex programmes have access to all data that is required. In both programmes the normal vector, in combination with the tangent and the binormal vector, is used to calculate the appropriate transformation matrix that is then applied to the surface-detail normal in the fragment operations.

For reflections four texture shader stages are setup to perform a transformation of each detail normal from surface-local space to world coordinates. The normalization of the transformed normal is achieved by a lookup in a normalisation cube map. The texture stages also compute the reflection vector based on the transformed normal. This reflection vector points into the environment map to obtain the reflection colour. A register combiner setup evaluates the Fresnel approximation based on the normalised normal and stores the result in the α portion of the frame buffer.

For refractions the normals of the height-field are calculated identically to the reflection case, but the transformation matrix is set to the matrix \mathbf{M} . The texture shaders transform the normal from surface-local space into the approximate transmission vec-

tor with respect to world coordinates. The transmission colour is obtained by a texture lookup in the environment map. Since all texture stages are already consumed, the surface normal vector cannot be transformed. Therefore the lighting calculation has to take place in surface-local space, utilising the register combiners.

Fig. 3 shows the results of different rendering effects. Fig. 3 (a) displays reflections only. Image (b) shows refractions of the fluid surface. The fabric on the pool ground appears distorted through the waves on the fluid surface. The Fresnel blending in (c) clearly exhibits that, in the area close to the viewer, one can only see the refractions while further away from the viewer the reflections play a major role. Fine details of the fluid surface, introduced by the per fragment surface detail, provides a natural appearance of the surface, as shown in (d).

In order to visualise the flow of the fluid, particles and other objects can be inserted in the fluid. In Fig. 4 the stream is visualised through leaves floating in the water. Fig. 5 shows an example for the coupling between flow and wave excitation. Fluid is streaming into the simulation grid from the top-left border. Behind the rectangular pillar the flow exhibits the typical vortices forming the wake behind an obstacle in the flow. Further material in the form of videos is available for download¹.

The achieved frame rates depend on the CPU for the simulation part and on the GPU for the rendering part. Table 1 shows results for an Athlon 1200 system with a GeForce3 graphics card. All test scenes were rendered in full quality. The bottom row of the table represents the mere rendering performance without any simulation. The case where the CPU tends to be the limiting factor can be seen in the first row. Here the frame rates increase only a few frames by decreasing the percentage of the displayed fluid. Grids about the size 64^2 provide a good trade-off between the load of the CPU and the load of the GPU.

8 Conclusion and Future Work

We have introduced a novel technique for real-time simulation and rendering of fluids for computer graphics applications. Combining a Navier-Stokes based 2D flow computation with a wave propagation technique on a height-field, very fast simula-

¹<http://www.vis.informatik.uni-stuttgart.de/~weiskopf/fluid>

simulation grid size	screen area of displayed fluid				
	100%	75%	50%	25%	0%
128x128	15	16	16	17	18
64x64	40	45	52	61	62
16x16	50	56	70	99	130
none	50	56	70	100	131

Table 1: Performance (fps) for an Athlon 1200 with a GeForce3 at a resolution of 800×600 pixels.

tions of realistic looking fluid behaviour are possible. Our method is well suited for modelling gentle motions of water in rivers or lakes. The high-quality rendering of the fluid surface supports reflections, refractions and Fresnel blending at fragment level. We have presented a method to exploit current consumer-level graphics hardware to evaluate these effects completely within the GPU. Since our hardware requirements are extremely low, we serve a wide range of systems and provide a very good compromise of speed versus quality and portability.

To further enhance the visual appearance of the rendered fluid surface, other properties of fluid light interaction could be considered. This includes, e.g., adding caustics to the rendering using a simplified model such as by Stam [17] which is also appropriate for consumer-level graphics hardware.

Additionally several enhancements are possible for the simulation part. E.g., the coupling between flow and wave propagation can be extended by advecting the height values with the flow. This would produce a more realistic transport of waves. Similarly the motion of the surface detail texture should be affected by the velocity field. Texture advection could be used to implement this feature on graphics hardware.

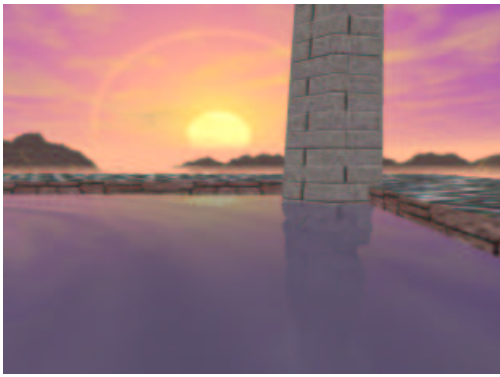
Also an approach to automatically sample the depth of the fluid from the surrounding scene geometry should be investigated. In the rendering part the obtained depth information will help to attenuate the light through the fluid, leading to a more sophisticated lighting model. The simulation part could also take advantage of these automatically generated depth information. For example the refraction of waves due to the real shape of the underlying scene geometry could be modelled.

References

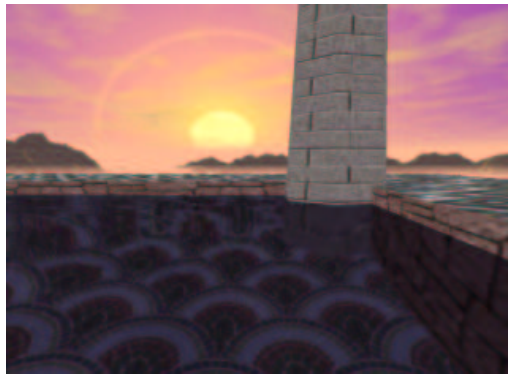
[1] M. Abbot and D. Basco. *Computational Fluid Dynamics – An Introduction For Engineers*. Longman Science & Tech-

nology, 1989.

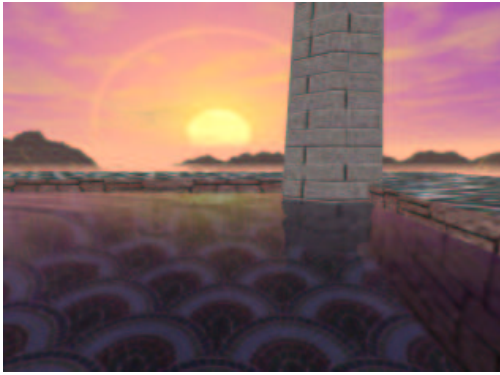
- [2] J. X. Chen and N. D. V. Lobo. Toward interactive-rate simulation of fluids with moving obstacles using Navier-Stokes equations. *Graphical models and image processing: GMIP*, 57(2):107–116, 1995.
- [3] A. J. Chorin. Numerical solution of the Navier-Stokes equations. *Mathematics of Computation*, 22(104):745–762, 1968.
- [4] D. P. Enright, S. R. Marschner, and R. P. Fedkiw. Animation and rendering of complex water surfaces. In *SIGGRAPH 2002 Conference Proceedings*, pages 736–744, 2002.
- [5] C. Everitt. Mathematics of per-pixel lighting. <http://developer.nvidia.com>, 2001. NVidia Corporation.
- [6] N. Foster and D. Metaxas. Realistic animation of liquids. In *Graphics Interface*, pages 204–212, 1996.
- [7] A. Fournier and W. T. Reeves. A simple model of ocean waves. In *SIGGRAPH 1986 Conference Proceedings*, pages 75–84, 1986.
- [8] M. Griebel, T. Dornseifer, and T. Neunhoffer. *Numerical Simulation in Fluid Dynamics – A Practical Introduction*. SIAM, Philadelphia, 1994.
- [9] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189, 1965.
- [10] D. Hinsinger, F. Neyret, and M.-P. Cani. Interactive animation of ocean waves. In *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation*, pages 161–166, 2002.
- [11] K. Iwasaki, Y. Dobashi, and T. Nishita. Efficient rendering of optical effects within water using graphics hardware. In *Proceedings of Pacific Graphics 2001*, pages 374–383, 2001.
- [12] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH 1990 Conference Proceedings*, pages 49–57, 1990.
- [13] G. A. Mastin, P. A. Watterberg, and J. F. Mareda. Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Applications*, 7(3):16–23, 1987.
- [14] D. R. Peachey. Modeling waves and surf. In *SIGGRAPH 1986 Conference Proceedings*, pages 65–74, 1986.
- [15] K. Perlin. An image synthesizer. In *SIGGRAPH 1985 Conference Proceedings*, pages 287–296, 1985.
- [16] J. Schneider and R. Westermann. Towards real-time visual simulation of water surfaces. In *Proceedings of the Vision, Modeling, and Visualization Conference 2001 (VMV-01)*, pages 211–218, 2001.
- [17] J. Stam. Random caustics: Natural textures and wave theory revisited. In *SIGGRAPH 1996 Visual Proceedings*, page 151, 1996. Technical Sketch SIGGRAPH'96.
- [18] J. Stam. Stable fluids. In *SIGGRAPH 1999 Conference Proceedings*, pages 121–128, 1999.
- [19] J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. In *SIGGRAPH 1993 Conference Proceedings*, pages 369–376, 1993.
- [20] P. Y. Ts'o and B. A. Barsky. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics*, 6(3):191–214, 1987.
- [21] G. D. Yngve, J. F. O'Brien, and J. K. Hodgins. Animating explosions. In *SIGGRAPH 2000 Conference Proceedings*, pages 29–36, 2000.
- [22] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, New York, 1971.



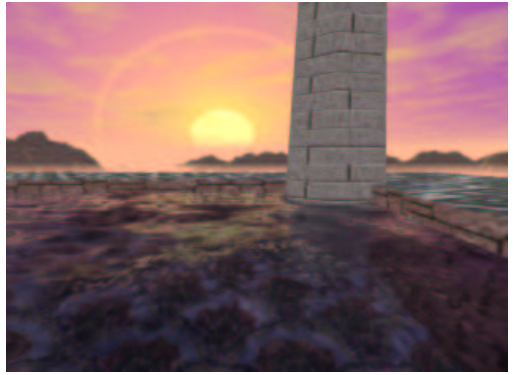
(a)



(b)



(c)



(d)

Figure 3: Comparison of rendering effects. Image (a) shows reflections only, (b) refractions only, (c) Fresnel blending of reflections and refractions, (d) all rendering effects including a noise-based surface detail.

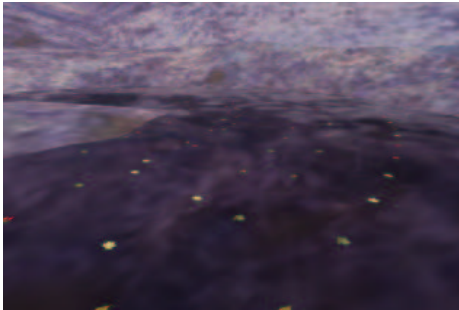


Figure 4: Visualisation of the stream through leaves in the fluid.

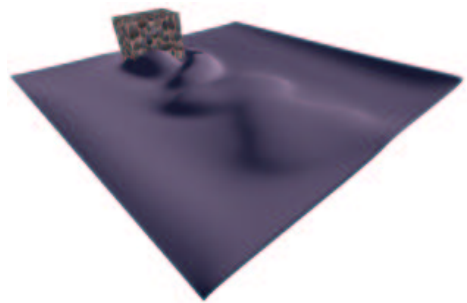


Figure 5: The typical wake, forming behind an obstacle in the flow.

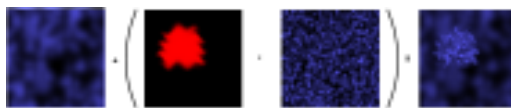


Figure 6: Modulating the surface detail with a local effects mask.