Nathan Holmberg
2514355

Modeling of Turbulent Water over Natural Terrain

Project Report

Abstract

Water phenomena are some of the most visually spectacular effects found in nature. The purpose of this project is to try to model flowing water in rivers, over waterfalls etc with the intent that the model can be used as part of a larger environment or scene. The model presented here uses hydrostatic theory to incorporate a 2D height field and particle system to model respectively the main volume and spray of turbulent water. The user is able to submit any environment formed from spheres and panels making the solution very flexible and adaptable. While the representation used here provides only an indication of the achievable effects and the current results are not yet rendered photo-realistically, the way in which the data from each frame is stored means that integration with a third party ray tracer would not be overly difficult. As would be expected the efficiency of this model depends on the complexity of the scene and while real time speeds are rarely possible the model is not prohibitively slow and compares favorably with alternative efforts.

Introduction

The complexity and power of water flow in nature is both impressive and beautiful. It is also a part of our everyday lives and so well known to the human eye that unrealistic motion is easily discernable.

It is no surprise therefore that much effort has been applied in computer graphics to try to capture water in a convincing way. This is far from trivial as most phenomena can be attributed to complex molecular interactions that occur trillions of times a second, and as such are beyond the modelling power of today's computers. Instead it is the goal of computer graphics to produce models with an underlying basis in physical principles that create, as accurately as possible, a large scale approximation of these local interactions.

As research in this area has evolved models have moved from only being able to represent certain effects, such as the motion of deep water waves to the exclusion of all else, to more general models with a firmer basis in hydrodynamics capable of realistic motion that follows expected behaviour in a range of situations. These more recent models allow animators to specify environment and start conditions and the model will do the rest.

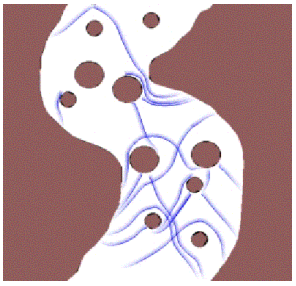
The purpose of this work is to build a general model but with a focus on the natural movement of rivers, rapids and waterfalls. As such turbulence, spray, and the like should be accounted for while not being treated as special cases.

This report describes a model obtained through the emulation and extension of previous work in this area. While by no means a complete or error free, our model has been able to produce results that show its potential.

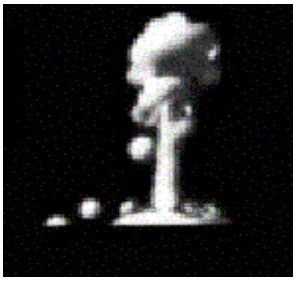
Previous Work

The earliest work on modelling water in computer graphics used mathematical models and explicit functions to model surface behaviour. Some, while obeying physical equations, were very inflexible outside their intended scope such as the early modelling of deep sea waves by Schachter [12] while others tried to model surface phenomena in 2D velocity fields [7] (Figure 1: a).

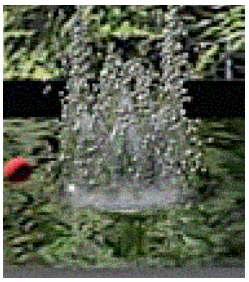
Rudimentary particle systems were the next method used. Initially particle interaction was ignored and particles simply bounced around the environment [11] [13]. This produced reasonable effects for waterfalls and other instances where particle movement had enough energy to break the molecular level bonds as needed. In particle systems equations for interaction can be complex and computationally expensive as each particle may be affected by every other i.e. a computational complexity of $O(n^2)$ which is inhibitivly slow when dealing with hundreds of thousands of particles. However, without inter-particle forces volumes of water can not be modelled well and the usually identifiable effects of adhesion and cohesion are missing. Some of the work at modelling these effects includes Miller and Pearce's [8] connected particle system which used forces to imitate soft collisions based on the difference in particle's positions (Figure 1: b). This worked well for viscous fluids, as was their intention, but is unsuitable for the number of particles or scale necessary to model large bodies of water.



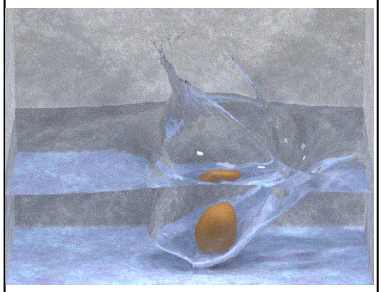
a) The shockwaves of water passing rocks [7].



b) A fountain of water [8].



c) Splash caused by ball hitting water [10].



d) Splash caused by ball hitting water [4].

Figure 1. Examples from previous works. Full images can be found in their respective papers

The paper by Kass & Miller [6] is often attributed as being the first in the development of the column system. This used simplified flow equations, based on hydrostatics, between columns and was treated as a 2D height field. This implicitly allowed for the modelling of surface phenomena such as waves and could efficiently model large bodies of water. Further work by O'Brien & Hodgins [10] developed this model by allowing for the interaction by external objects and splashes (Figure 1: c). At this stage a column's height was still represented by one height variable meaning that vertical isotropy was assumed and only very simple (flat) environments could be used. Mould & Yang [9] furthered this work by dividing each column into a user defined number of cells, relaxing the assumption of vertical isotropy. Another improvement was made by allowing for more complex environments. Even with the use of cells, however, column systems are still only a two and a half dimensional model at best and many effects such as the curling of waves cannot be reproduced. There has also been some work at applying the 2D version of the Navier-Stokes equations to column systems [3].

The final and most recently developed method of simulation uses the full Navier-Stokes equations to compute motion in a 3D grid of voxels, examples of which can be found in Enright et al[4] and Foster & Fedwik [5]. This is a very expensive solution but also creates the most realistic effects as can be seen in Figure 1: d. These systems work by calculating the velocity of flow out of the 6 faces of each voxel using the pressure, temperature etc of each voxel. By calculating which voxels contain water, object, or air, the environment can be rendered for extremely realistic animation. This method has been used in movies such as Shrek but its difficulty is perhaps illustrated in Jeffrey Katzenberg's statement that the pouring of milk into a glass was the hardest shot in the movie [4][5]. This approach does have its limitations of course with perhaps the largest being the loss of information within cells, meaning that the grid size plays an overly important role in the accuracy of the model.

Model

After testing a particle only model, logical as particles can model three dimensional behaviour which turbulence invariably is, it became apparent that the computational time for such a system was prohibitive for large bodies of water. Instead by using different systems for spray and volume, as suggested by several sources [10] [9], a faster and more believable model could be created.

Volume Model

Continuing the work of O'Brien & Hodges and Mould & Yang [10] [9] the volume model, representing the main body of water in our model, is a column based system. Using columns holds the advantages of easy surface creation as the top of all columns are known and less flow calculations needed so the system is less computationally intensive.

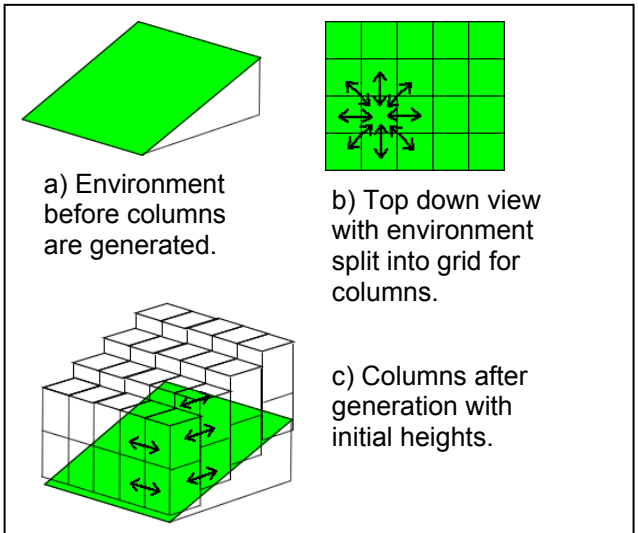


Figure 2. Column Generation from environment. Arrows indicate position of some pipes.

The structure of the volume model is much the same as that used in the previous work mentioned above. That is the environment is divided into equal sized squares which form the base of columns as in figure 2. All columns start with a user defined height which then varies over time dependant on the calculated flows. Source / sink columns are the only ones to retain their heights allowing for in and out flows to the system. Pipes are then created between each of the eight adjacent columns and each cell that could overlap during the course of the simulation. Pipes are also created between the cells of one column and the air above the adjacent columns. At this point the system is ready to begin simulation.

The underlying basis for the flow equations that are used in this simulation is the science of hydrostatics, or that describing the pressure of fluids at rest. The equations related with this approach are simple, both to understand and to compute, and as a result are easy to implement.

For any column in the grid the hydrostatic pressure can be calculated from the equation below based on the work of Bernoulli.

$$Q = h\rho g + \frac{1}{2}\rho v^2 + (p_0 + E)$$

Where Q is the total pressure, h the height of the column, g acceleration due to gravity, ρ the density of the fluid, v the velocity of flow. p_0 the air pressure and E the pressure arising from external forces which together form the pressure energy term. In this case the height of the column is the height above some arbitrary point in the world so long as the same point is used for all columns.

Using the pressure differences between cells it is possible to calculate the acceleration and from that the flow that should occur between cells. The final equation for the flow velocity (η) through a pipe is:

$$\eta = f\eta_0 + \Delta t\left(\frac{Q_{head} - Q_{tail}}{\rho l}\right)$$

Where l is the pipe length, f is a friction coefficient (as suggested in [9]), and η_0 is the flow in the previous time step. An interesting point to note is the lack of any viscosity parameter in this equation. This is because one of Bernoulli's assumptions was that the distances between points of measurement were so small that viscous losses were negligible, instead the friction coefficient used allows energy to slowly escape from the system. While not physically justified it serves as

an ad-hoc method of including viscosity and in all examples was set at 0.995.
Using the flow calculated for the pipe the volume of water that should be moved through it is calculated by:

$$V = \Delta t \eta c$$

Where *c* is the cross-sectional area of the pipe, or the amount of overlap between the cells. Because mass is to be conserved the volume removed from one column is the same as that added to the other. Care must also be taken not to allow a volume of less than zero to occur.
This system is very fast considering the mass of water that is being represented but there are several problems.
Columns pose a problem as a representation because turbulence is a three dimensional feature; this is considered one of the classical characteristics of turbulence [1]. While using the 'cells' given by Mould & Yang relaxes the assumption of vertical isotropy this goes a long way to explaining some of the model's inability to simulate certain situations. A second problem arises from the fact that turbulence is a feature of flow and not of the fluid 'at rest'. This means that while hydrostatics may be easy to use the equations generated for flow are incomplete and ignore many of the visible characteristics of water such as shear stresses. This is perhaps the largest flaw in the model and something that lends itself to further research.

Spray Model
The spray model is used to model water as it breaks free of the main volume of water. There is no easy physical solution to when spray should be created and as such assumptions must be made instead. Earlier work with column systems were concerned mainly with generating splashes from hitting objects and as such used vertical velocity thresholds for generating spray. Because the idea here is to model waterfalls the assumptions used here are those regarding the heights of wave crests before they become unstable (when the wave height is 0.78 of the water depth) [15]. This obviously allows waterfalls to form easily but also works for rapids as large flow velocities form 'spikes' of water that while erroneous are then turned into spray due to their large heights.
The spray system begins its evolution when particles are generated. First the number of particles (or volume) needs to be determined. Using the formula to calculate flow through a weir it is possible to determine the required flow rate and therefore volume. This is through the equation [2]:

$$flowrate = \frac{2}{3} B H^{\frac{3}{2}} \sqrt{2g}$$

Where *B* is the base length, *H* the height and *g* the acceleration due to gravity. The volume to pass through in this time step can then be found by *flow rate * time step*. This determines the number of particles to be created as all those generated, except the last, is of a user defined volume. The last of course needs to be the remainder of volume to be moved as mass must be conserved and not created. Depending on the scale and resolution of the model currently being used this can be set to achieve the best looking results. The position of the particles is also easily determined and is set at a random position in the face that the particle is being generated from. The final initial variable that is needed for each particle is velocity and this can also be generated from the flow rate equation above. In this case the velocity of flow through a column's face is found to be flow rate / face area. Flows within the column structure may mean the velocity should not be perpendicular to the front of the face; to account for this the average flow from surrounding columns is used to give direction to the scalar velocity calculated above.
In many cases the velocity imparted to the particle should not only be horizontal but also include an initial y velocity. To do this the difference in total height between the column for which particles are being generated and the column behind is used in the classic formula

$$v^2 = u^2 + 2as$$

Where *v* is the current velocity, *u* is the initial velocity, *a* the acceleration, in this case gravity, and *s* the distance covered. While this may not be a perfect physical solution it does manage to provide a more believable representation.
One of the methods that were initially considered to help create the illusion of water pooling and incompressibility was the use of cohesion. This involves creating small forces between water

particles which attract and repel neighbours in the effort of keeping an optimal distance apart and is an effort to model intermolecular bonds. After both observing the inefficiency and inaccuracy of using cohesion within the particle system and after reading enough to convince that such was unneeded due to water's low viscosity so long as the movement was turbulent, such as is the case with spray, [13][14] this was excluded when the particle and column systems were combined. The visual effects of such cohesion might be achieved by appropriate rendering of the particle system but due to time constraints we were not able to explore this topic in more detail. This means that the only active force during a time step is gravity.
Collision detection with the boundary is the same as for the original particle system described in the implementation section and so collisions with columns is the only thing explained here.
After calculating which column the particle is above (or within), the particle's y coordinate value is tested to see if it should be absorbed into the column. Simply increasing the column's volume and destroying the particle was found not to be accurate enough however as this would force the column's height up, often absorbing more particles in the process. This would not have been a problem for the small scale splash effects that the underlying model was designed for but causes considerable problems when the particles are modeling waterfalls where there are many particles hitting at any one time. After trying to spread the volume of a colliding particle over several columns it was found that by instead modeling the force of impact and subsequent pressure increase in the column not only was the problem reduced but more realistic effects were generated. The equations used were the same as for external objects colliding with the water presented in Mould and Yang. The force on the object is given as two terms.

$$F = -v\mu - V\rho g$$

Where v is the velocity of the object, μ is the viscosity, V the volume of water displaced, ρ the density of the fluid and g the acceleration due to gravity. The first term describes the force of the fluid on the particle and the second is the force due to buoyancy. Because the force on the fluid must be equal but oppositely orientated to that on the water droplet this formula can then be used to determine the force on the column. Using the formulas:

$$P = m / A \qquad \text{and} \qquad m = \frac{F}{a}$$

We can then calculate the resultant pressure of this force on the column, stopping it from rising unrealistically.

Implementation

Several elements had to be created for the implementation of the model described above that are not explicitly explained. This section describes some of the methods used to allow the model to function.

Solver

The first thing that was required to be implemented was a solver for computing a numeric solution of an ordinary differential equation (ODE), as opposed to the more commonly taught symbolic solutions. The main idea is to discretize the independent time variable of the ODE and to approximate derivatives with finite differences. The basis for this implementation is the design described by Witkin & Baraff [16] and the three methods used are those laid out in their paper. By following this structure it became very easy to integrate the solver with other systems later on. The three numerical solution methods approximate the solution at the next time step by using the Taylor expansion:

$$x(t_0 + h) = \sum_{k=0}^{\infty} \frac{x^{(k)}}{k!} h^k$$

The main differences being the number of terms considered for approximation.
The simplest and fastest algorithm within a given time step is the Euler method as it only accounts for the first two terms i.e.

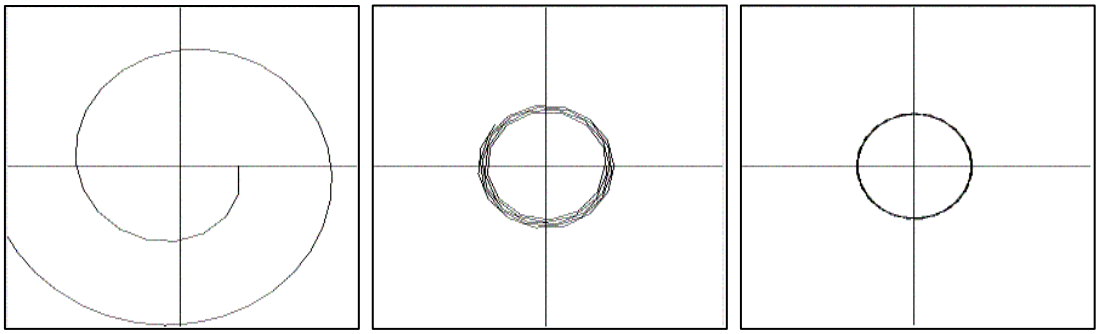
$$x(t_0 + h) = x_0 + h\dot{x}(t_0) + O(h^2)$$

$O(h^2)$ in this case represents the error inherent in the solution. The second implemented is the

Midpoint method which includes one more term of the series and a higher accuracy.

$$x(t_0 + h) = x_0 + h\dot{x}(t_0) + \frac{h^2}{2}\ddot{x}(t_0) + O(h^3)$$

The final and most accurate method implemented is Runge-Kutta of order 4(RK4) and an error of $O(h^5)$. This includes a further two terms of the series (not written here).



a) Euler method b) Midpoint method c) Runge-Kutta 4 method

Figure 3. Illustration of different solver methods and their (in-) accuracies. All three attempt to draw a circle of $x^2 + y^2 = 1$

As would be expected as accuracy increases so does the computation time needed. Dependant on its use, however, much larger time steps can be taken while remaining relatively accurate. The accuracy of the three methods can be seen in Figure 3.

Particle System

A basic particle system was written next. Particles have a position, velocity, and mass but no volume. They can be affected by forces, either form outside the system or from other particles and can represent anything from rigid structures to fluids (as described earlier).

Based on the outline given by Witkin [17] the system that was written was as general as possible to allow for extension later. All forces and environmental constraints were represented by interfaces that could be implemented as required. The system itself implements the interface required by the solver and as such represents itself as a point moving through 6-n space where n is the number of particles and the position and velocity of each particles is represented by a 6-dimensional. The derivative of each particle's concatenated 6 vector [x1,x2,x3,v1,v2,v3] is represented by [v1,v2,v3,f1/m,f2/m,f3/m] where f is the force acting on the particle.

By representing the system as a point in 6n-dim space, all forces between particles can be applied simultaneously so that the system stays consistent for each time.

Solving for a particular time step begins with the system calculating the initial forces acting on the particles. The particle system can then pass itself to the solver method of choice. Because the solver views f(x,t) as a black box (i.e. doesn't know how the function is evaluated) it calls back for solutions at intermediately positions. At these points the system recalculates the forces and responds appropriately.

After the new positions and velocities are determined collision detection with the environment is performed. Certain panels defined in the environment file are used as boundary conditions to stop particles escaping. While any parallelogram is allowed as a panel it is important to note that the collision detection used here is only valid for rectangles. To find if a particle has passed one of these panels first a check to see which side of the panel the particle is on through the use of the formula:

$$(P - V_1) \bullet n \leq 0$$

Where *P* is the particles position, V_1 is one of the vertices of the panel and the unit normal of the plane on which the panel rests. A further check is necessary to see if the particle also lies within the rectangle described by the panel. If the four conditions below hold true then the point is within

the parallelogram.

$$(P - V_2) \bullet (V_2 - V_1) < 0$$
$$(P - V_2) \bullet (V_2 - V_3) < 0$$
$$(P - V_4) \bullet (V_4 - V_1) < 0$$
$$(P - V_4) \bullet (V_4 - V_3) < 0$$

Where *P* is the particle's position and V_1 through V_4 are the four vertices of the rectangle in anticlockwise order. This is a simplification of the equation to check which side of a line a point is. As the sign is the only thing we are interested in it is more efficient to disregard excess calculation.

A final check is made to ensure the particle is not 'legally' on the other side of the panel. By tracing the particles path back one time step and checking that the old position was 'correct' it can be assumed that the particle has indeed hit the panel.

As described in the model section particles also have to be tested as to whether they have hit the main volume of water. Because columns are stored in a 2D array in their physical order it is very easy to determine which column a particle is directly above. The system's minimum x and z points are stored as global variables so it is simply a matter of finding the difference between the particle's position on the x/z plane and the system minimums and scaling by the resolution. I.e.

$$index_i = (P_i - \min_i) \times resolution$$

Where *index* is the array index, *P* the particle's position, *min* the minimum value for the system and *resolution* the number of columns per unit length. If the particle is outside the bounds of the column array or its vertical position (y coordinate value) is below all columns in the system the particle is destroyed. Otherwise it is absorbed into the volume model as described above.

Finally each particle is checked to ensure it hasn't exceeded its lifetime after which the particle system clock is increased by the time step given and the system is again deemed stable.

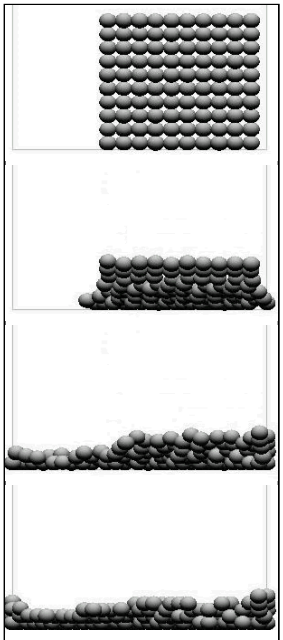


Figure 4. Frames showing cohesion in the particle (spray) system. See video file for more detail.

Initially, when the particle system was being used exclusively, it was considered appropriate to try to model cohesion between the particles to mimic molecular interaction. The first attempt was to apply a simple spring with a limited area of effect. This spring forced particles towards an equilibrium situation. This then evolved through the application of the work done by Miller & Pearce [8] to that shown in the figure. These forces were not found to be appropriate however as they made the fluid appear too viscous to be believable for water. They were also very slow.

Column System

As discussed above the volume model follows that used by Mould & Yang [9] closely. Before the system may be used, however, the columns must be generated from the environment specified and pipes between all adjacent cells created.

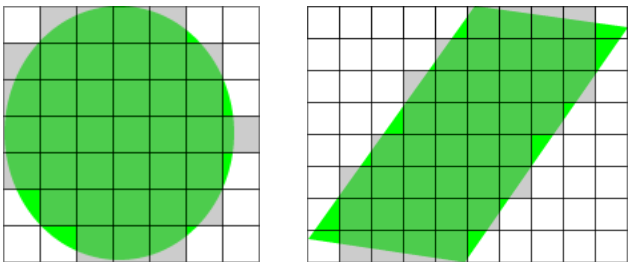


Figure 5. Creating columns from the environment file. Shaded squares indicate a column to be created

Using the same environment file format as that originally used for the particle system, each shape is cycled through checking if columns should be created. Spheres, used to represent rocks and obstacles are the first. For each sphere in the environment a cross-section along

the x/z plane is used. Taking the lower left corner of a bounding rectangle each possible column position is checked to determine if it is within the cross-section. For this an approximation is used where each corner is checked to see if it is inside, if two or more corners are inside then a column is created in that position. The base height of each column created is found by determining the height of the center as it would be projected onto the sphere.

Each panel is also cycled through to create columns although, unlike spheres, each panel can serve one of three different purposes. If the area, on the x/z plane, is zero and the panel is not a source panel it is treated as a boundary panel for particle collision detection and is stored for use as such. Source panels also have no area but columns are still created along the line described by the first and second vertices. These columns have an initial height equal to the difference between the first and third vertices' y values which is kept regardless of outflows. However, most panels act in much the same way as all spheres by providing the ground environment for the column system.

Once again finding the bounding rectangle into which the projection of the panel onto the x/z plane fits, each possible column position is checked as to if a column should be created there. To do this opposite corners of the column position are tested as to whether they are left of the lines formed by V_1 and V_2, V_2 and V_3, V_3 and V_4, and V_4 and V_1. Each time a score is incremented for each corner to the left of each line, if both corners are outside a line then no column is created otherwise if the 'score' is above six (there was a maximum of two lines any corner was outside of) then a column is created as appropriate. This method is only an approximation but serves well enough.

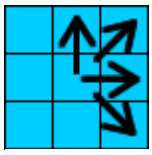


Figure 6. Direction of pipe creation.

After each shape has been cycled through we now have an unordered array of columns. By sorting them into a 2D array so they are placed according to their physical position a much better representation of the environment can be made, and more efficient algorithms can be used later. If there are two columns for any position then either the higher or source column is taken.

Because the columns are stored in order the creation of pipes is rudimentary. Each column is cycled through creating pipes in the directions shown in the figure. As described in the model section all possible pipes are created and then checked for validity later instead of dynamically creating pipes in each time step.

To increment the system each pipe is cycled through with the intention of calculating flow. If a pipe's cross-section is less than 0 (i.e. the two cells don't overlap) nothing occurs, for those that do the flow and volume to be moved are calculated and the cells are updated accordingly.

While the user can choose the amount by which the system increments between each frame the system is hard coded to increment by only 0.005. Instead the number of smaller time steps needed is taken until the system has incremented by the amount required.

Representation

As can be seen from the example pictures a suitable rendering method for this particular model has not yet been developed. Instead columns, particles and the environment are represented using OpenGL. In this case particles are represented by spheres which have the same volume as the particle is supposed to have. To try to give an indication of the distance and time a particle has traveled each particle's colour is calculated based on how long it has been alive, starting blue and turning white over several seconds. The Environment is also very simple with spheres and quads making the rocks and ground. Columns are represented by a series of quad strips between the centers of columns as shown in the figure. Each vertex also needs a normal which is found by the cross product of two sides of the current quad. If a large vertical jump or a column which has no height is encountered the quad strip is halted and begun anew after the jump.

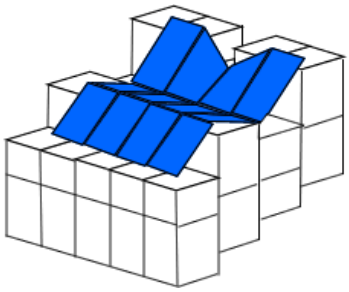


Figure 7. How the surface is generated from the columns.

Conclusion / Future Work

Using hydrostatics and drawing on information from fluid dynamics the model created for this project is capable of creating recognizable effects for the simulation of rivers and waterfalls. The speed of the model depends largely on the number of columns needed to cover the area and the number of particles in the scene. For example the simple river scene depicted in figure C ran at 1 frame (at 15 frames a second) every 15 seconds with 13000 columns, each with 4 cells and just over 3700 particles. Conversely the forked waterfall in figure B took one and a half minutes per frame and had 27500 columns, each with 4 cells and 110000 particles.
The model as written here has been shown to represent some river situations well but also suffers from some notable fallacies. Perhaps the largest of these is the lack of viscous shear forces which is needed for creating eddies and visual effects such as shockwaves around rocks and near the edges of the river. A second effect that this model fails to model well but could quite easily is spray from water particles hitting columns too hard. In reality this often causes a semi permanent area of white water and mist around the base of the waterfall.
Also so far this model is only suitable for representing the river and the location of whitewater etc. Only with the addition of rendering could it be used for anything else.

References:

[1] Abbot M & Basco D. *Computational Fluid Dynamics – An Introduction for Engineers*. Longman Scientific & Technical.

[2] Badger W & Banchero J. *Introduction to Chemical Engineering*. McGraw-Hill.

[3] Chen J & Lobo N. *Towards Interactive-rate Simulation of Fluids with Moving Obstacles by Navier-Stokes Equations*. CVGIP: Graphical Models and Image Processing V. 54 n 3. 1995

[4] Enright D, Marschner S & Fedwik R. Animation and Rendering of Complex Water Surfaces. SIGGRAPH '02:736-744 2002

[5] Foster & Fedwik. Practical Animation of Liquids. SIGGRAPH '01:23-30

[6] Kass M & Miller G. *Rapid, Stable Fluid Dynamics for Computer Graphics*. Proceedings of SIGGRAPH '90 V. 24 n 4:49-57

[7] Neyret F & Praizelin N. *Phenomenological Simulation of Brooks*. Computer Animation and Simulation Sep 2001:53-64

[8] Miller G & Pearce A. *Globular Dynamics: A Connected Particle System for Animating Viscous Fluids*. Computer & Graphics V. 13 n 3:305-309 1989

[9] Mould D & Yang Y. *Modeling water for computer graphics*. Computer & Graphics V. 21 n 6:801-814 1997

[10] O'Brien J & Hodgins J. *Dynamic Simulation of splashing fluids*. Computer Animation '95:198-205

[11] Reeves W. *Particle Systems – A Technique for Modeling a Class of Fuzzy Objects*. Computer Graphics V. 20 n 4:65-74. 1986

[12] Schachter BJ. *Long crested wave models*. Computer Graphics and Image Processing V.12:187-201 1980

[13] Sims K. *Particle Animation and rendering using data parallel computation*. SIGGRAPH '90

[14] Stein C & Max N. *A Particle-Based Model for Water Simulation*. Prepared for SIGGRAPH '98

[15] Thorton E & Guza R. *Energy Saturation and Phase Speeds Measured on a Natural Beach*. Journal of Geophysical Research V. 87 c 12:9499-9508 1982

[16] Witkin A & Baraff D. *Differential Equation Basics*. SIGGRAPH '94

[17] Witkin A. *Particle System Dynamics*. SIGGRAPH '94

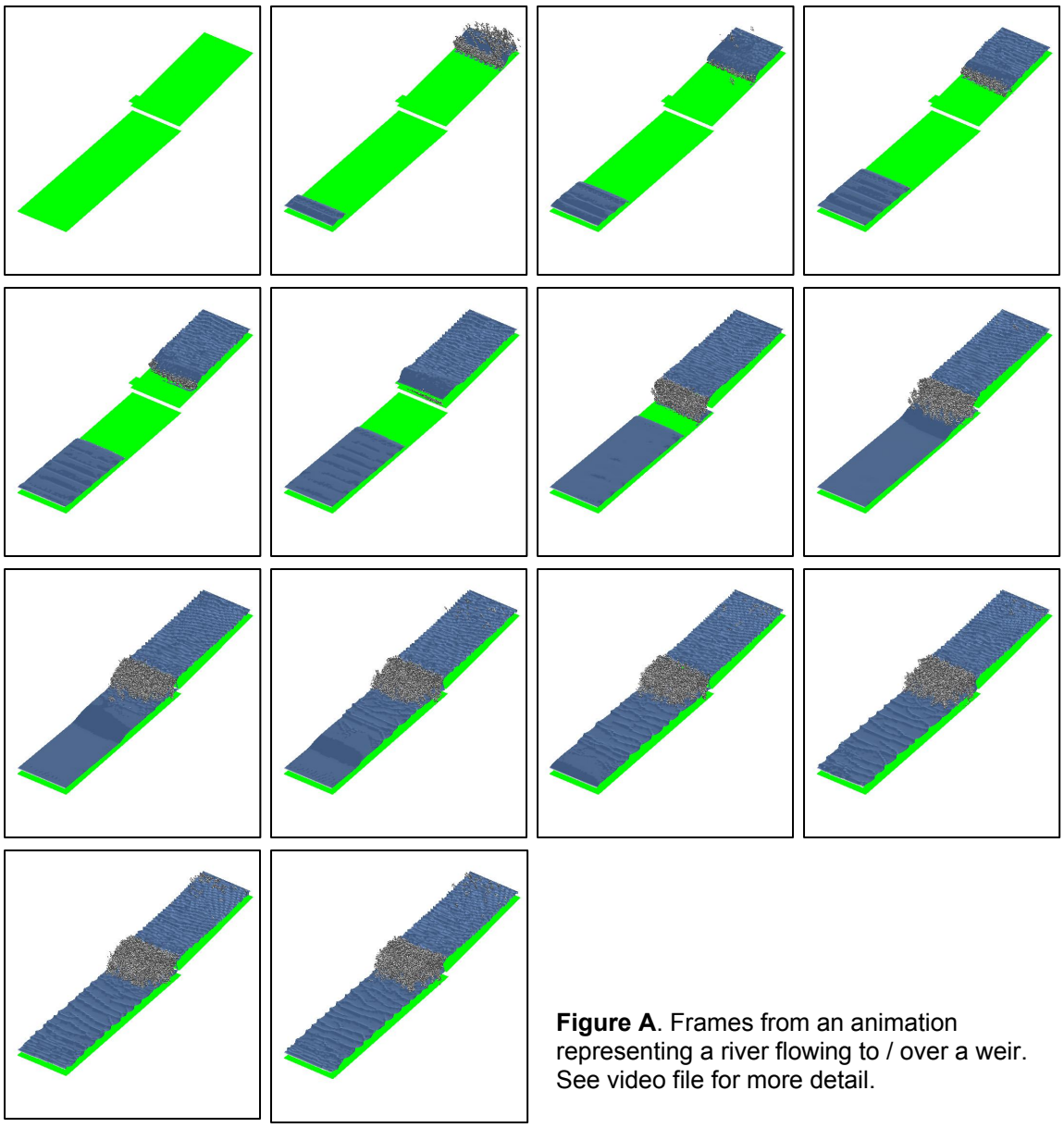


Figure A. Frames from an animation representing a river flowing to / over a weir. See video file for more detail.

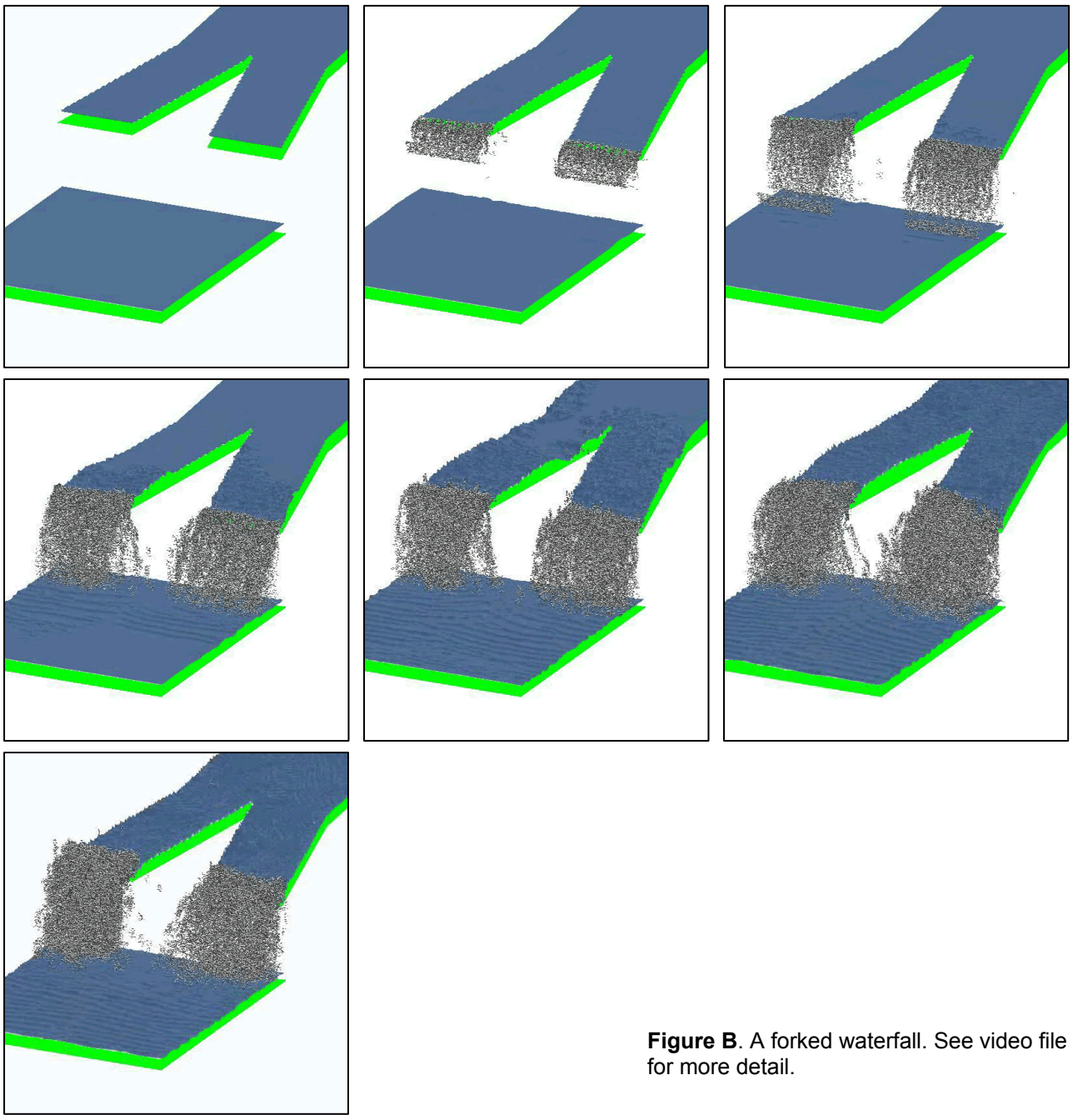


Figure B. A forked waterfall. See video file for more detail.

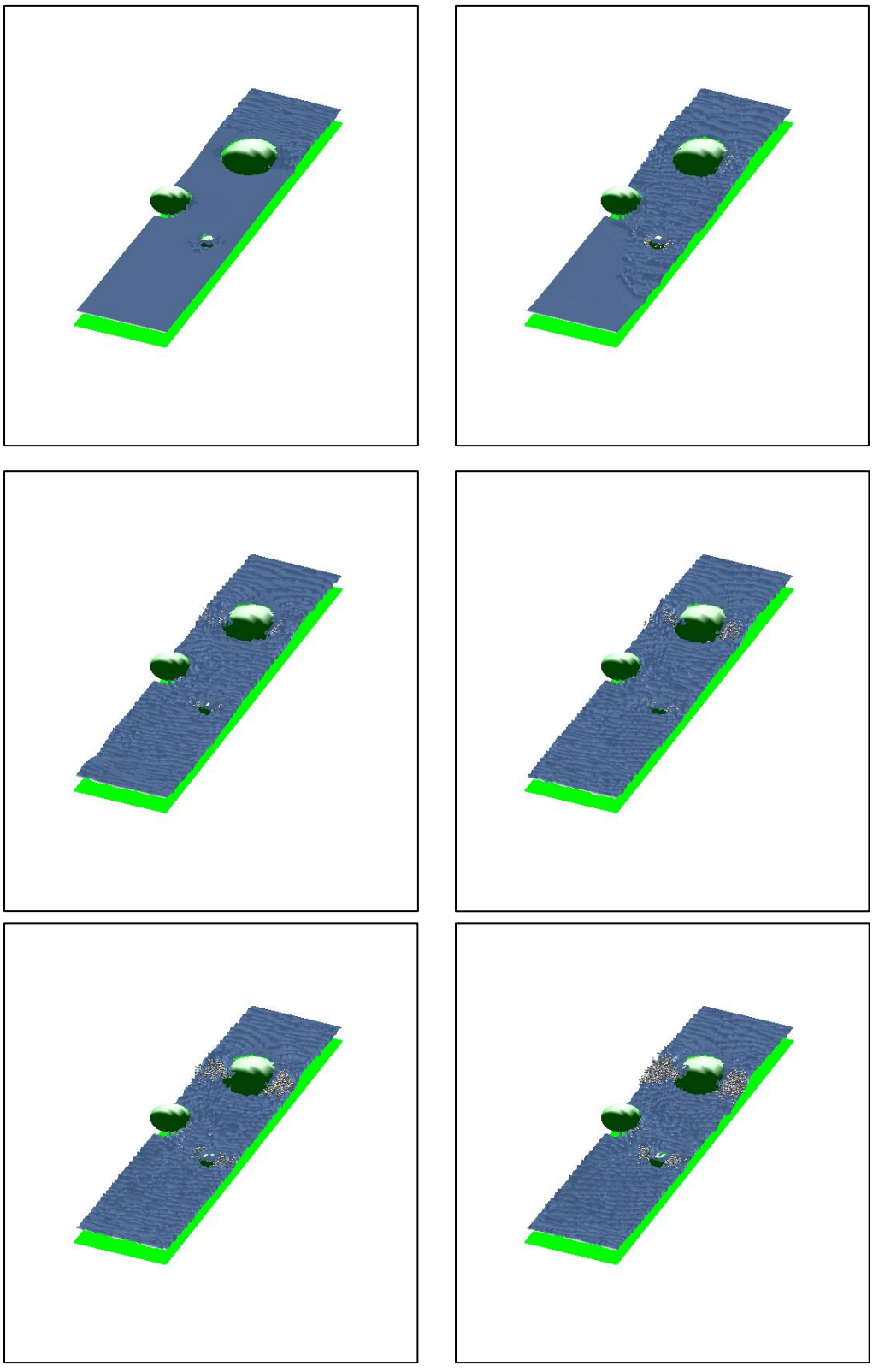


Figure C. A Simple river with rapids around the large rock. See video file for more detail