Combining Regional Time Stepping With Two-Scale PCISPH Method

Joel Begnert
Rasmus Tilljander

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden
This thesis is submitted to the Department of Computer Science & Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Authors:
Joel Begnert
joby10@student.bth.se

Rasmus Tilljander
rati10@student.bth.se

University advisor:
Dr. Prashant Goswami
Dept. Computer Science & Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden
Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

Context. In computer graphics, realistic looking fluid is often desired. Simulating realistic fluids is a time consuming and computationally expensive task, therefore, much research has been devoted to reducing the simulation time while maintaining the realism. Two of the more recent optimization algorithms within particle based simulations are two-scale simulation and regional time stepping (RTS). Both of them are based on the predictive-corrective incompressible smoothed particle hydrodynamics (PCISPH) algorithm.

Objectives. These algorithms improve on two separate aspects of PCISPH, two-scale simulation reduces the number of particles and RTS focuses computational power on regions of the fluid where it is most needed. In this paper we have developed and investigated the performance of an algorithm combining them, utilizing both optimizations.

Methods. We implemented both of the base algorithms, as well as PCISPH, before combining them. Therefore we had equal conditions for all algorithms when we performed our experiments, which consisted of measuring the time it took to run each algorithm in three different scene configurations.

Results. Results showed that our combined algorithm on average was faster than the other three algorithms. However, our implementation of two-scale simulation gave results inconsistent with the original paper, showing a slower time than even PCISPH. This invalidates the results for our combined algorithm since it utilizes the same implementation.

Conclusions. We see that our combined algorithm has potential to speed up fluid simulations, but since the two-scale implementation was incorrect, our results are inconclusive.

Keywords: smooth particle hydrodynamics, two-scale, regional time stepping, fluid simulation
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Complex scene with user defined H-region</td>
<td>9</td>
</tr>
<tr>
<td>4.2</td>
<td>Two-scale H-region boundary handling</td>
<td>11</td>
</tr>
<tr>
<td>4.3</td>
<td>RTS on Double dam break</td>
<td>13</td>
</tr>
<tr>
<td>4.4</td>
<td>Region expansion for $R_u$</td>
<td>14</td>
</tr>
<tr>
<td>4.5</td>
<td>Correction scheme for RTS</td>
<td>15</td>
</tr>
<tr>
<td>6.1</td>
<td>Visual comparison of the algorithms</td>
<td>22</td>
</tr>
<tr>
<td>6.2</td>
<td>Lower time step along the border</td>
<td>24</td>
</tr>
</tbody>
</table>
List of Tables

6.1 Simulation results from the Gallery scene ........................................ 21
6.2 Simulation results from the Double dam break with pillars scene .......... 23
6.3 Time distribution over all parts in the combined algorithm .................. 23
6.4 Minor step time details ................................................................. 23
6.5 Simulation results from the Double dam break scene ......................... 24
# List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SPH</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Two-Scale simulation</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>RTS for PCISPH</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Update validity</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Density Correction RTS</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Combined Technique</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>Pre-Minor step</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>Minor Step</td>
<td>18</td>
</tr>
</tbody>
</table>
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Fluid simulation</td>
<td>1</td>
</tr>
<tr>
<td><strong>2 Background and Related Work</strong></td>
<td>3</td>
</tr>
<tr>
<td>2.1 Smooth Particle Hydrodynamics</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td>4</td>
</tr>
<tr>
<td><strong>3 Method</strong></td>
<td>6</td>
</tr>
<tr>
<td>3.1 Aim and Objectives</td>
<td>6</td>
</tr>
<tr>
<td>3.2 Research Questions</td>
<td>6</td>
</tr>
<tr>
<td>3.3 Method</td>
<td>6</td>
</tr>
<tr>
<td>3.3.1 Implementation</td>
<td>7</td>
</tr>
<tr>
<td>3.3.2 Experiment</td>
<td>7</td>
</tr>
<tr>
<td><strong>4 Base Algorithms</strong></td>
<td>8</td>
</tr>
<tr>
<td>4.1 Two-scale simulation</td>
<td>8</td>
</tr>
<tr>
<td>4.2 Regional time stepping</td>
<td>12</td>
</tr>
<tr>
<td><strong>5 Combined Algorithm Overview</strong></td>
<td>17</td>
</tr>
<tr>
<td>5.1 Our combined method</td>
<td>17</td>
</tr>
<tr>
<td>5.1.1 Implementation differences</td>
<td>19</td>
</tr>
<tr>
<td><strong>6 Results and Discussion</strong></td>
<td>21</td>
</tr>
<tr>
<td>6.1 Results</td>
<td>21</td>
</tr>
<tr>
<td>6.2 Discussion</td>
<td>21</td>
</tr>
<tr>
<td><strong>7 Conclusions and Future Work</strong></td>
<td>25</td>
</tr>
<tr>
<td>7.1 Future work</td>
<td>25</td>
</tr>
<tr>
<td><strong>Appendices</strong></td>
<td>30</td>
</tr>
<tr>
<td>A Goswami and Battys unpublished paper</td>
<td>31</td>
</tr>
</tbody>
</table>
Introduction

1.1 Fluid simulation

In the field of computer graphics, realism has always been desired. When it comes to fluid simulation there are multiple approaches to achieve realistic liquids. For the most part, the time cost associated with a realistic simulation is high. Therefore, research has been devoted to reduce total computational time. Methods for doing so are usually divided into three subgroups: Eulerian, Lagrangian, and a combination of the two.

Eulerian, or grid-based, methods is one alternative for simulating fluids in the computer graphics industry, it has high coherence with the ground reality. In Eulerian methods physical quantities (pressure, density, velocity, etc.) of the fluid are defined on a grid and are then changed over time, but the grid remains fixed. This can be visualized as observing a river pass by while sitting at the riverbank; properties of constant points in the water change continuously, but the observed position remains the same.

Lagrangian methods, as opposed to the Eulerian, move fluid mass around explicitly. The quantities are tied to a small part of the fluid which is tracked through the whole simulation. An analogy would be sitting in a boat and drifting down a river while watching the water around the boat. These methods are usually more computationally expensive for higher details because a parcel of fluid needs to be directly aware of its surroundings which requires expensive neighborhood calculations. On the other hand, they offer advantages on simulating small scale features like droplets and splashes, because the surface is not bound to a grid but is instead represented by particles on the surface. In addition, Lagrangian methods conserve mass implicitly and do not need a separate scheme for mass conservation, because each particle represents a unit of mass and all particles are always tracked.

Combinations of both Eulerian and Lagrangian methods utilizes both approaches using, for instance, a grid in one part of the fluid, typically the bottom, and particles for the surface.

Whether a simulation method is Eulerian or Lagrangian, it is progressed with a time step, a small value which is classically chosen globally and do not vary
throughout the simulation. The time step determines how far the simulation is
progressed in each iteration. A larger time step forwards the flow of the fluid
further, and thus decreases the required total computational time, since fewer
animation frames are performed to reach the same simulated time. Although a
large time step seems desirable, choosing it too large could, for instance, cause
two particles in the fluid to be progressed in such a way that they occupy the
same space. This can cause physically incorrect movement and displeasing visual
results and is referred to as instability or unstable simulation [Ihmsen et al.,
2014b]. Visually displeasing results means that explosions or waves occur where
there should be none, while in a good visual result the fluid behaves as it would
in real life.

It is, however, still desirable to have as large as possible value for the time
step but still maintain a stable simulation. While the total computational time
changes, the time for each iteration stays the same. This implies that a simulation
that uses a large time step can be simulated further with the same computational
cost compared to the same simulation using a lower time step. Such a decrease
in computational time is desirable and is what many techniques for simulating
fluid seek to achieve, which will be made clear in the next chapter.
Chapter 2

Background and Related Work

2.1 Smooth Particle Hydrodynamics

Much of today’s work is based on a variant of a Lagrangian method called Smooth Particle Hydrodynamics (SPH) [Lucy, 1977; Gingold and Monaghan, 1977], as the name implies SPH uses particles to simulate fluids. Most calculations are based on a particles neighborhood, i.e. all particles within a certain distance of the particle. Each particle uses the distance to its neighbors to calculate a density value, this value is then used to compute the forward force and velocity. The new position is calculated by forwarding every particle’s current position a small distance, using its velocity and a given time step. See algorithm 1.

Algorithm 1 SPH

1: while animating do
2:     for each particle \( i \) do
3:         find neighborhoods \( N_i(t) \)
4:     for each particle \( i \) do
5:         compute density \( \rho_i(t) \)
6:         compute pressure \( p_i(t) \)
7:     for each particle \( i \) do
8:         compute forces \( F_{p,v,g,ext}(t) \)
9:     for each particle \( i \) do
10:         compute new velocity \( \mathbf{v}_i(t+1) \)
11:         compute new position \( \mathbf{x}_i(t+1) \)

It is important to make sure that particles maintain physically correct placement, failure to do so can lead to undesirable visual effects and may even introduce instability. This placement restriction is called incompressibility and most liquids in real life are incompressible. The standard SPH method [Lucy, 1977; Gingold and Monaghan, 1977] does not enforce incompressibility so in an attempt to do so, Becker and Teschner [2007] introduced Weakly Compressible SPH (WCSPH). While this method enforced near-incompressibility it came with a time step restriction which required a low time step to stay stable.
To increase the time step, Solenthaler and Pajarola [2009] developed a Predictive-Corrective Incompressible SPH (PCISPH). By using larger time step their approach could simulate further with less total computational time. This, however, introduced aforementioned errors in density and made the entire simulation unstable. They solved this by introducing an additional step in the algorithm which predicts each particle’s new position and iteratively applies a correction force if the density is over a certain threshold. The correction force is then used to calculate the particle’s actual new position. Although this leads to each iteration being more expensive when compared to WCSPH, a significant speedup is achieved because the time step can be many times larger.

To further decrease total computation time, Solenthaler and Gross [2011] also introduced the Two-scale particle simulation method. This method uses two separate simulation with different particle sizes. The particles in one simulation only interact indirectly with the particles in the other. One simulation uses larger particles, and therefore a smaller amount of them, and the other simulation uses smaller particles but in a limited area. This area is a subset of the whole simulation where details are more important. Since the rest of the simulation requires fewer particles, and therefore less computations, a speedup is achieved.

In contrast to reducing the number of particles, Regional Time Stepping (RTS) by Goswami and Batty [2014] reduces computation time by using varying time steps based on the amount of movement in each area. Particles in areas with small movements are assigned larger time step and are therefore not required to update as often. On the other hand, particles in areas with much movement are assigned a smaller time step. Thus less computational time is spent on the parts that exhibits less complexity.

Seeing as the two-scale particle simulation and regional time stepping methods improve on two different aspects of the PCISPH technique, we propose a method combining the two. Our method, like two scale, uses two simulations both of which individually calculate what time steps to use in the different regions, the same way Goswami and Batty [2014] does.

### Related Work

With Eulerian techniques, it is common to use height-field models to simulate water, introduced by Kass and Miller [1990]. This was later combined with marked particles to keep track of the free surface Foster and Metaxas [1996]. A method called the level set method uses several sets of levels and surface marker particles Enright et al. [2002].

The Lagrangian method SPH was originally used for fluids other than incompressible liquids: stars and particle clouds in astrophysics Lucy [1977], Gingold and Monaghan [1977]; smoke, gas and fires Stam and Fiume [1995]; deformable bodies Desbrun and Gascuel [1996]. The SPH method was first applied to incom-
pressible flows by Monaghan [1994] and later by Müller et al. [2003] to animate fluids.

Usually, one of two strategies is used to enforce incompressibility: incompressible SPH (ISPH) or weakly compressible SPH (WCPSH), Becker and Teschner [2007]. The ISPH methods solve a pressure Poisson equation while WCPSH uses a stiff equation of state (EOS). To reach density fluctuations below 1%, a common limit for incompressibility, the stiffness value for the EOS has to be large enough. However, larger stiffness values requires smaller time steps as the value often dominates the Courant-Friedrichs-Levy condition Solenthaler and Pajarola [2009]. In contrast, ISPH methods can have higher time step but each iteration takes much longer time to compute. This was adressed by He et al. [2012] in the local Poisson SPH and by Ihmsen et al. [2014a] in the Implicit Incompressible SPH.

WCSPH has been the basis for many algorithms trying to speed up the fluid simulation. Goswami et al. [2010] used the processing power of the GPU and Ihmsen et al. [2011] examined possibilities and suitable data structures for parallel implementations.

Other attempts have been made to improve on the SPH method, most notably the PCISPH method Solenthaler and Pajarola [2009], mentioned before. Approaches to further speed up the simulations include varying the particle size to reduce the total number of particles. This can be done dynamically over the whole fluid Adams et al. [2007]; Hong et al. [2008] or only in certain areas Solenthaler and Gross [2011] Horvath and Solenthaler [2013].

There has also been research in adaptively varying the time step to allow for longer time steps when shorter ones are not necessary. Both globally adaptive Ihmsen et al. [2010] Goswami and Pajarola [2011] or locally adaptive Goswami and Batty [2014] time stepping has shown increase in efficiency.

For a more thorough discussion of different implementations, we refer to Ihmsen et al. [2014b].
Chapter 3

Method

3.1 Aim and Objectives

The aim of this thesis is to explore the possibility of combining two methods designed to improve on two different aspects of PCISPH. The goal is to reduce total computation time and ultimately speed up the process of creating realistic fluids in computer graphics. To this end, we have defined two research questions.

3.2 Research Questions

RQ1. Is it possible to construct a combined algorithm based on the regional time stepping, which introduces an elevating effect on the time step, together with the two-scale simulation, which lowers particle count?

RQ2. If a combined algorithm is possible, will it simulate large body of fluids faster compared to its two base techniques and PCISPH?

3.3 Method

In order to answer the research questions we used implementation and experiment methodologies. These methods suit this kind of issue since we get measurable results from experiments, which lead us to implement the new algorithm to be able to perform experiments.

We implemented both the two-scale algorithm and the regional time stepping algorithm, followed by the construction and implementation of the combined algorithm. We evaluated the first question using two criteria: is the combined algorithm able to run a simulation and if so, does the visual results differ significantly from the original algorithms?

The second question is tightly coupled to the first one, and to answer it we defined a set of test scenarios. In these scenarios we changed particle number and the scene configuration. Tests were then performed using all four algorithms with
same, or equivalent, parameters on all three scenes. We measure the time it took to run each test and then calculated and compare the speed up of all algorithms over PCISPH.

3.3.1 Implementation

By implementing both base algorithms we got a foundation, as well as a deeper understanding to help us construct our combined algorithm. During our implementation we used the visual C++ 12.0 compiler on Windows 8.1 OS and compiled towards 64-bit.

3.3.2 Experiment

Our experiment consisted of twelve tests, each running one simulation until 10 seconds of real time had been simulated. Throughout the tests we varied the scene complexity and particle count, however, particle size and time step remained the same: 0.06 and 0.001s respectively. The scenes used were inspired by scenes in both original papers, in that they were relatively simple with a few obstacles.

The tests were performed on computers with 16 gigabyte of memory, Intel Core i7-2700K processors (3.5 GHz, 8 cores) and were multithreaded using OpenMP. In all tests we used static particles for walls and collision objects for the simplicity when it comes to implementation. Due to time limitations, we could only run each test once.
Chapter 4

Base Algorithms

Our algorithm is based on the two algorithms: two-scale Simulation [Solenthaler and Gross, 2011] and regional time stepping [Goswami and Batty, 2014]. However, Goswami and Batty [2014] limited the research to WCSPH, while our algorithm only uses PCISPH. In an unpublished paper by Goswami and Batty, the RTS method is expanded for use with PCISPH. This is what we base our algorithm on. In this chapter we will explain these methods and why they improve the simulation times. More details can be found in the respective original papers, the unpublished RTS paper can be found in its fullness in Appendix A.

4.1 Two-scale simulation

The two-scale simulation technique, outlined in algorithm 2, uses two separate simulations to simulate a body of fluid, one high and one low resolution. The low resolution simulation, L-simulation or simply L, uses large particles and represents the entire simulation. The high resolution, H-simulation or H, on the other hand uses smaller particles and exist only as a subset of the L-simulation.

By using larger, and thus fewer, particles for the entire simulation, as opposed to an equivalent simulation containing only high resolution particles, the total computational time is reduced. By identifying the regions where large particles fail to achieve the desired level of detail, we can make use of smaller particles to maintain visually pleasing results. Since the resolution of the particles in those regions is higher than the rest of the fluid, they are are called H-regions.

Identifying an H-region can be done with multiple approaches. It can be a predetermined region where a user has observed complex movement, like obstacles where collisions occur, see figure 4.1. Another option is to use dynamic approaches, such as using the view frustum, i.e. use larger particles for the part of the simulation that is currently not viewed by the observer. A third approach assumes that the visual results is most critical at the surface, which can be determined as the H-region using surface detection [Solenthaler et al., 2007] and flood fill algorithm to expand downward from the surface.

Each particle in the H-simulation, referred to as an H-particle, has $1/n$ the diameter of a particle in L, L-particle, where $n$ is the resolution factor and a
Figure 4.1: A complex scene with a user defined H-region. The red particles represents the boundary region between the L and the H-region. Notice the particles in L (left) are larger than in H (right). The gray particles are static particles.

multiple of 2. Due to instability risks introduced by using smaller particles which can more easily end up in physically incorrect positions, the H-simulation requires \( 1/n \) the time step of the L-simulation. To prevent the H-simulation from lagging behind, we perform \( n \) physics calculations and position forwarding for H every time step.

Because the L-simulation acts as a base simulation and the H-simulation is an independent simulation, they need to be coupled so that the resulting fluids match. To tie the simulations together, a parent and children relationship is
used. An H-particle has exactly one parent particle from L, which is computed by finding the closest L-particle. In contrast, an L-particle can have an unlimited number of children, but can only act as a parent as long as it occupies a position within the H-region. Only L-particles may act as a parent, and only H-particles may be created as children.

When updating an H-particles parent and since a particle does not move very far each frame, checking its current parents neighbor for a closer particle typically suffices. If all particles in the neighborhood are farther away from the H-particle than a distance of an L-particles diameter, the whole L-simulation is searched for the closest particle.

Algorithm 2 Two-Scale simulation

1: while animating do
2:   compute physics for L
3:   determine regions in L
4:   transfer region information from L onto H
5:   add / delete $H_{\text{boundary}}$ particles
6:   interpolate quantities from L onto $H_{\text{boundary, activeRelax}}$
7:   for $n\text{Substeps}$ do
8:     compute physics for H
9:     advect $H_{\text{boundary}}$
10:    update parent particle in H
11:    interpolate feedback information from $H_{\text{active}}$ onto L

A small area surrounding the H-region is defined as the boundary region. The boundary region is a critical part of the simulation because it is the only region where H-particles can be deleted or created. When an L-particle enters the boundary region from L, child particles are created to represent the parent within H. In contrast, if a parent leaves the boundary region and enters the L-region it deletes all associated children, see figure 4.2.

It is important that all particles has a correct neighborhood for the physics calculations to be performed correctly. Since the H-particles are created at the edge of the H-region, they only have neighbors on one side and will be pushed out again. To counter this, the particles in the boundary region do not calculate neighborhoods or any physics. Instead the H-particles within the boundary region are advected by the flow of L until they are deleted or the parent enters the H-region. This way the H-particles in the H-region has a satisfying neighborhood along the boundary without losing the flow of the fluid.

Furthermore, when a parent particle enters the H-region from the boundary region, its children enter what is called a relaxation period. This period exists because the particles coming from the boundary region may lie in a configuration which could cause sudden changes in density should they start physics calculations
Figure 4.2: L-particles entering the boundary region of the H-region creates child particles in the H-simulation. These particles are advected until their parent exits the boundary region, they are then either deleted or set to the relaxed state, depending on the direction of the parent.

immediately. Therefore, the relaxed particles interpolate their quantities from advected values to physically correct values over a short period of time.

The L-particles in the active region receive a feedback force from the smaller particles so that L and H do not diverge significantly. In addition, simulations with larger particles typically experience more dampening effects which is mitigated by the feedback from the smaller particles in the H-region. The feedback force is calculated from the average velocity of all children to each L-particle in the H-region.
Chapter 4. Base Algorithms

4.2 Regional time stepping

In conventional SPH methods, all regions of fluid are integrated with the same time step, even though different regions could actually be updated at different frequency depending on the velocities and forces in each region. With the assumption that small parts of a fluid are subject to approximately the same forces, Goswami and Batty discretizes the whole fluid domain into blocks. Each side of a block is twice as long as a particles diameter, thus a block can contain only a small number of particles. By giving each block a specific activity level based on the forces and velocities of the containing particles, areas of the fluid with more rapid movement can be given more computational power (i.e. more calculations per animation frame) and regions with less movement can be given less, thus decreasing the total computational time.

The activity levels are called regions and there are $n$ of them, $R_1$ to $R_n$, where $R_1$ is the region with highest force or velocity, as figure 4.3 shows. A block is assigned the highest activity level, the lowest region number, for which it satisfies three equations based on the contained particles’ maximum velocity, force, and the time step. For more detail about these equations we refer to Goswami and Batty [2014].

One animation frame is called a major step and it consists of $n$ minor steps, where $n$ is the number of activity levels. Each minor step forwards the simulation $\Delta t_b$ seconds, the base time step, thus each major step is $n\Delta t_b$ seconds long. The idea is that some computations only need to be calculated once every major step, instead of every minor step. The neighborhood grid update, usually one of the most expensive parts of the PCISPH algorithm, is an example Solenthaler and Gross [2011].

Particles in $R_1$ and $R_2$ could have such velocities that they travel outside their blocks over one time step, therefore, those regions are expanded 4 layers of blocks.

In addition, blocks bordering higher activity level blocks are marked as observed blocks, $R_{ob}$, which are checked for density errors whenever the bordering region is corrected, as can be seen in figure 4.4. This way we only have to check the density of blocks neighboring the erroneous blocks since they are the only ones in risk of becoming erroneous themselves.

Inside the minor steps, all physics computations is made, however, since slow moving particles probably have the same neighborhood for an entire major step it does not need to be re-computed every minor step. On the other hand, faster moving particles will need to keep their neighborhood and densities updated. We keep track of which particles should be updated using the variables compute and validity for each particle. Before the minor steps we set compute to true and validity to the particles activity level. The validity variable is decremented each minor step and compute is updated according to algorithm 4. In algorithm 3, conditional means that a particle will only update if its compute variable is set to true.
Figure 4.3: RTS simulation on the scene Double dam break. The regions are color coded as follows: $\mathcal{R}_1$ red, $\mathcal{R}_2$ green, $\mathcal{R}_3$ blue, and $\mathcal{R}_4$ yellow. $\mathcal{R}_1$ occurs just as the particles are hitting the wall since much force is exerted on them from the wall. In contrast, the particles in the opposite corner belongs to $\mathcal{R}_4$ since they are practically still.

Seeing as the regions with low velocity and forces do not change the particle density rapidly, the number of iterations they require in the correction loop is lower than the amount the higher activity levels need. Therefore, the particles are corrected according to algorithm $[5]$ and the scheme in figure $[4.5]$, where $e$ denotes blocks that could not be corrected previous iteration, and will be corrected regardless of region membership.

If some density error remains after the first 3 iterations, local corrections are
Figure 4.4: We use the same idea with region expansion and observed blocks \( R_{ob} \) as in Goswami's extended paper, we mark blocks around erroneous blocks and region borders as observed.

performed on the blocks still marked as erroneous. If the average density error of the whole fluid is larger than a threshold, however, additional corrections are performed on all blocks. Should the correction loop fail to converge for a total of 6 iterations, the time step for the major steps is reduced to half, that is \( n/2 \Delta t_b \), because the neighbor grid update will happen more often and the simulation can therefore handle sudden density changes better. The time step is changed back to \( n\Delta t_b \) if the simulation has run 10 major steps with all correction loops exiting with every blocks density corrected.
Algorithm 3 RTS for PCISPH

1: while animating do
2:   update block neighborhood grid
3:   find max velocity \( v \) and force \( F \) per block
4:   calculate block activity level
5:   expand region \( R_1, R_2 \)
6:   find observed block \( R_{ob} \)
7:   for \( nRTSMinorSteps \) do
8:     conditional update neighborhood
9:     conditional update density \( \rho \)
10:    conditional update external forces \( F^{ext} \)
11:    reset pressure \( p \) and pressure forces \( F^p \)
12:    prediction correction step (Algorithm 5)
13:    logic for half/full major step
14:    update velocity \( v \) and position \( x \)
15:    update validity (Algorithm 4)

Figure 4.5: The density correction schedule used defined by Goswami and Batty in their unpublished paper. When read from left to right one can view the four minor steps, read from top and down we see the iteration within each minor step. In the first iteration within the first minor step all four regions \( R_1 \) \( \ldots \) \( R_4 \) performs prediction correction steps. It is designed so that each region performs at least one density correction each minor step. After the first iteration within a minor step we perform an iteration for \( R_x \) if \( x \) modulus of the current minor step equals zero. However, \( R_4 \) is omitted in the fourth minor step as it made no impact on the visual results. Regions with fast moving particles receive more correction each minor step.
Algorithm 4 Update validity
1: for all $i \in \text{particles}$ do
2:    decrement validity
3:    if validity$_i \leq 0$ then
4:       set compute$_i = \text{true}$
5:    update validity$_i$
6: else
7:       compute$_i = \text{false}$

Algorithm 5 Density Correction RTS
1: while iter $\leq \text{minIterations} \lor \text{density}_{\text{error}} \geq \text{max density}_{\text{error}}$ do
2:    predict velocity $\mathbf{v}$ and position $\mathbf{x}$
3:    predict density $\rho$ and pressure $p$
4:    if density error $\rho_{\text{err}}$ remains outside active regions then
5:       update $\mathcal{R}_{\text{err}}$ and $\mathcal{R}_{\text{ob}}$
6:    if particle has turn then
7:       compute pressure force $F_p$
8:    iter++
9:    if $\rho_{\text{err}} \geq \text{max} \rho_{\text{err}}$ \& iter $\geq 3$ then
10:       minIterations++
11:    if $\rho_{\text{err}} \geq \text{density threshold}$ then
12:       perform extra global corrections
13:    else
14:       perform local corrections on $\mathcal{R}_{\text{err}}$
Chapter 5

Combined Algorithm Overview

5.1 Our combined method

For our combined algorithm we had to find a way to fit the two-scale and RTS algorithms together. By an iterative process we found that they could be combined by computing region membership for blocks in both the L and the H-simulation. Since both simulations share many algorithm steps, we decided to encapsulate some parts of the RTS algorithm into two steps: pre-minor step and minor step, resulting in algorithm 6. The pre-minor step contains calculations that are only required once per major step, for instance updating neighborhood grid and determining region membership, this can be seen in algorithm 7. After performing the pre-minor step for the L-simulation we determine where the H-region is and add or remove H-particles, depending on their parents’ position. The pre-minor step for H is performed in a likewise manner to L and then the minor step loop is entered.

Algorithm 6 Combined Technique

1: while animating do
2:   pre-minor step for L
3:   determine H-region
4:   add/remove H-particles
5:   pre-minor step for H
6:   for nRTSMinorSteps do
7:     minor step for L
8:     interpolate quantities
9:   for nTwoScaleSubSteps do
10:     minor step for H
11:     advect boundary
12:     interpolate feedback info
13:     update parent particle

The minor step contains most things needed to calculate new positions for all particles. A few steps are computed in both the pre-minor step and the minor
Algorithm 7 Pre-Minor step
1: update neighborhood grid
2: for each particle $i$ do
3: compute density $\rho_i$
4: for each particle $i$ do
5: compute external forces $F_{i}^{ext}$
6: find max velocity $v$ and force $F$ per block
7: calculate block activity level
8: Expand $R_1, R_2$
9: find observed blocks $R_o$

step, these steps can thus be omitted the first time they are encountered in the minor loop. This can be seen in algorithm 8.

Because of the way $\Delta t_H$ and $\Delta t_L$ correlates (see section 4.1) the minor steps for H is performed $n_{TwoScaleSubSteps}$ times for every minor step for L. The conditional steps are only skipped the first time.

Algorithm 8 Minor Step
1: if $n_{RTSMinorSteps} \neq 1$ then
2: conditional update neighborhood
3: set max velocity $v$ and force $F$ per block
4: upgrade block activity level
5: for each particle $i$ do
6: conditional update density $\rho_i$
7: for each particle $i$ do
8: conditional update external forces $F_{i}^{ext}$
9: for each particle $i$ do
10: reset pressure $p_i$ and pressure forces $F_{i}^{p}$
11: prediction correction step
12: for each particle $i$ do
13: update velocity $v_i$ and position $x_i$
14: update validity (Algorithm 4)
15: cull observed blocks
16: cull error blocks

The prediction correction step has not been modified much compared to the one used in RTS, with the exception of handling relaxed particle. The relaxed particles are allowed a higher density error than the other H-particles, so we calculate separate max and average density error for particles in the relaxed state. Also, because of the nature of the relaxation interpolation, we do not expect relaxed particles to behave physically correct and thus they are not allowed to
mark their blocks as erroneous. However, blocks that contain relaxed particles may still be marked as erroneous by other H-particles in the same block. The block would be added to $\mathcal{R}_e$ and all particles in it, including the relaxed, would be further corrected.

After the prediction correction step we forward all L-particles using $\Delta t_b$ and update their corresponding validity variable. We also cull any blocks which were previously observed or marked as erroneous but no longer needs to be. Only blocks that were originally marked as observed blocks, the blocks found between borders, are kept between minor steps. All other observed blocks, which are marked by neighbouring erroneous blocks, are always reverted due to performance gain, as erroneous blocks might oscillate. If such observed blocks still have an erroneous block as a neighbor, they will be detected once again if required. We keep the originally observed blocks because the boundaries between the regions usually do not change inside a major step and it is unnecessary to recalculate the observed blocks every minor step.

When the L-simulation has executed the minor step we interpolate its particles’ quantities to use with the advection of the boundary particles in H. In our implementation the resolution factor, and therefore $n_{TwoScaleSubSteps}$, is set to 2, so we execute the minor step for H and advection twice. The minor step for H is performed in the same way as for L and the advection forwards every boundary particle in time using the interpolated values from their parents.

Interpolating feedback info and updating parent particle is performed in the same way as in the two-scale method. For the feedback info, each parent gets an average velocity from all its children and uses it to calculate a feedback force. For the parent update every H-particle receive its nearest L-particle as a parent.

The determination of the H-region and the adding and removing of particles is tightly coupled and since changing the number of particles requires the neighborhood grid to update, we chose to place those steps outside the minor loop. Parent update was placed outside for the same reason; if an H-particle gets a parent which is outside the H-region, the child would have to be removed. Only updating the parent particle every major step instead of every minor step gave us no stability issues or visual artifacts.

5.1.1 Implementation differences

In our preliminary tests we noticed that the RTS’ neighbor search for $\mathcal{R}_1$ particles was very expensive. We found that using the same neighbor search as for the other regions produced equally good visual results at a considerably shorter time. Therefore, we chose to not use the expanded neighbor search for neither RTS nor our combined method.

Furthermore, we found that our algorithm did not benefit from the local correction step used in RTS. And since no local corrections were performed, the major steps were never halved and thus we could omit the associated logic. This
gave us no discernible speedup but it simplified the algorithm. Based on empirical experiments, the constants used for our algorithm are the same as in the two-scale algorithm implementation; relax time, $\beta$-value for feedback force, and the higher allowed density error for the relaxed particles. However, for the pressure force interpolation we modified the velocity limitation; instead of limiting the final velocity of the particle, we only limit the velocity calculated by the pressure force that iteration. This gave us a much smoother transition from the relaxed state, with fewer sudden density changes. The $\beta$-value for the force interpolation was nevertheless experimentally set to 0.05, same as in the original two-scale paper [Solenthaler and Gross, 2011].
Chapter 6

Results and Discussion

6.1 Results

As can be seen in tables 6.1 through 6.2, our combined method has not shown the combined speedup of two-scale and RTS. In addition, our two-scale implementation has not given the speedup over PCISPH as shown in Solenthaler and Gross [2011]. It has, in some cases, even taken twice as long as PCISPH.

The RTS algorithm does not suffer from these slowdowns and show an increase in efficiency over PCISPH, as expected. We see that our combined algorithm gives a speedup over all other algorithms, except for in table 6.5 the double dam break scene, where RTS is slightly faster.

Visually, we see very few differences between the algorithms results, as is made clear in figure 6.1. There is, of course, some difference between the L-area in the two-scale and combined algorithms, and the corresponding area in PCISPH and RTS. However, that is to be expected from the two-scale base algorithm, since the particle size differs outside the H-region.

6.2 Discussion

The slowness of the two-scale simulation could be caused by a few differences from the original paper. Firstly, the resolution factor in our implementation is only 2, whereas in the two-scale paper it is in some tests set to 4. A resolution factor of 4 gives L-particles which are 64 times larger than the H-particles, reducing the total number of particles significantly. Secondly, our scenes and specifically

<table>
<thead>
<tr>
<th>Technique</th>
<th>Particle count</th>
<th>Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCISPH</td>
<td>1.1M</td>
<td>54343.5</td>
<td>-</td>
</tr>
<tr>
<td>Two-scale</td>
<td>L: 140k H: 400-600k</td>
<td>54923.5</td>
<td>0.989</td>
</tr>
<tr>
<td>RTS</td>
<td>1.1M</td>
<td>47750.7</td>
<td>1.138</td>
</tr>
<tr>
<td>Combined</td>
<td>L: 144k H: 400-600k</td>
<td>45385.8</td>
<td>1.197</td>
</tr>
</tbody>
</table>
the determination of the H-regions differs a bit. Our regions are up to 50% of the whole simulation domain, and Solenthalers and Gross’ seem to be around 25% in most of the scenes. Because of the amount of H-particles, the overhead from certain calculations (create/delete H-particles, boundary handling, parent update) might outweigh the benefits of the two-scale method.

However, the most likely cause of the result is the behavior of the relaxed particles. During our implementation we had some difficulties with the boundary and relaxed H-particles for the two-scale algorithm and our combined algorithm, since they use the same calculations. It is possible that there still exists some
Table 6.2: Simulation results from the Double dam break with pillars scene

<table>
<thead>
<tr>
<th>Technique</th>
<th>Particle count</th>
<th>Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCISPH</td>
<td>950k</td>
<td>47172</td>
<td>-</td>
</tr>
<tr>
<td>Two-scale</td>
<td>L: 120k H: 300-570k</td>
<td>117816</td>
<td>0.400</td>
</tr>
<tr>
<td>RTS</td>
<td>950k</td>
<td>41841</td>
<td>1.127</td>
</tr>
<tr>
<td>Combined</td>
<td>L: 120k H: 300-570k</td>
<td>40642.8</td>
<td>1.161</td>
</tr>
</tbody>
</table>

Table 6.3: Time distribution over all parts in the combined algorithm

<table>
<thead>
<tr>
<th>Pre-Minor L</th>
<th>Determine Region</th>
<th>Pre-Minor H</th>
<th>Minor steps</th>
<th>Update Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.30%</td>
<td>17.70%</td>
<td>7.40%</td>
<td>70.60%</td>
<td>3.00%</td>
</tr>
</tbody>
</table>

problems which causes the particles to get incorrect densities which in turn requires more iterations in the prediction-correction loop. The fact that most of the $\Phi_1$ regions exist near the border (as can be seen in figure 6.2) further supports this theory.

We can also see in table 6.3 and 6.4 that neither update parent nor feedback calculations, two-scale specific steps, are taking a lot of time. The determine region part alone takes a notable amount of time but this is likely due to bad memory management. Currently we de-allocate and allocate new memory every time we remove or create new H-particles. A better memory management scheme could reuse memory and thus alleviate the overhead of memory allocation and improve performance by reducing memory fragmentation. This is, however, probably not the cause of the two-scales slow results due to its significant difference compared to our combined algorithm, which uses the same memory management.

We see that our combined algorithm on average is faster than RTS and PCISPH, despite that the two-scale results shows much longer simulation time than PCISPH. This indicates that if our two-scale implementation gave the same speed up as in the original paper, we would see a large speed up in our combined algorithm as well.

One reason to why RTS is faster in the Double dam break scene could be that because there are no pillars in the H-region, fewer particles are determined to belong to $\Phi_1$, since less sudden impacts occur. This, in turn, could result in the

Table 6.4: Minor step time details

<table>
<thead>
<tr>
<th>L</th>
<th>H</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.90%</td>
<td>91.00%</td>
<td>1.10%</td>
</tr>
<tr>
<td>Update neighborhood</td>
<td>Update density</td>
<td>Prediction correction</td>
</tr>
<tr>
<td>36.60%</td>
<td>7.20%</td>
<td>46.00%</td>
</tr>
<tr>
<td>Technique</td>
<td>Particle count</td>
<td>Time (s)</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>PCISPH</td>
<td>950k</td>
<td>45486.4</td>
</tr>
<tr>
<td>Two-scale</td>
<td>L: 120k H: 300-500k</td>
<td>105931</td>
</tr>
<tr>
<td>RTS</td>
<td>950k</td>
<td>40519</td>
</tr>
<tr>
<td>Combined</td>
<td>L: 120k H: 300-500k</td>
<td>42342.4</td>
</tr>
</tbody>
</table>

particles getting slightly incorrect positions due to the reduction of calculations performed on them. The result would be that more iterations of the prediction correction loop would have to be performed, and as can be seen in 6.4 that is a large part of the algorithm. Thus, ultimately, making the combined simulation take more time than RTS.

Also worth noting is that each test was only performed once, and therefore there is a possibility that some results might have been compromised by external influences. However, this does not explain the long simulation time of the two-scale algorithm, since we see the same slowness for all three scenes.

Figure 6.2: The $\mathbb{R}_1$ area (colored red) is prominent along the border between H and L, even when the particles should not be experiencing large forces, like in the middle of the yellow $\mathbb{R}_4$ region.
We have presented a combined algorithm based on the two algorithms regional
time stepping and two-scale simulation. Seeing as we can run complete simula-
tions and that the visual results do not differ noticeably from the base algorithms,
the answer to our first research question is: it is possible to construct a combined
algorithm.

However, our implementation of the two-scale algorithm seems to give incor-
correct results and since our combined algorithm utilizes the same implementation
we can not be sure that our results are correct. Although we see a slight speed
up, we can not with certainty answer our second research question as the results
undoubtedly would change should the two-scale algorithm give better results.

To conclude, our combined method has potential, but it is restricted our two-
scale implementation. In addition, the speed up we already have makes up for
the algorithms relative complexity compared to RTS and two-scale simulation.

7.1 Future work

One continuation would be to try and reproduce Solenthaler and Gross’ results
with the two-scale implementation. This might be as simple as building better
suited scenes or implementing support for a resolution factor of 4, or it might
require a whole re-implementation of the algorithm. Regardless of the two-scale
implementation, the algorithm presented in this paper could also be expanded
using the GPU rather than the CPU. As mentioned by Goswami and Batty
[2014] the blocks can be treated as parallelization units for computing the physics
of particles within, as also implemented by Goswami et al. [2010].

To further explore the possibility to perform more complex calculations for de-
termining the high resolution area for particles in L could be interesting. In our al-
gorithm we manually determine, prior to the simulation, one single 3-dimensional
cuboid which we use to perform simple location based queries against. However,
as suggested by Solenthaler and Gross [2011], exploring the possibilities with ei-
ther surface detection or using a view frustum to define high resolution areas
could be an interesting next step.
There is another point of view that could be explored. Rather than combining the two methods as shown in this paper it could also be interesting to explore the possibility to take advantage of differently sized particles in different regions. For instance letting $R_1$ and $R_2$ be H-regions or even having one particle size per region level.

We would also like to run our tests several more times and perform statistical analyses on the results. This way we could establish the validity of our current results.


Appendices
Appendix A

Goswami and Battys unpublished paper
Asynchronous Liquids: Regional Time Stepping for Faster SPH
Prashant Goswami, Christopher Batty

Figure 1: Regional time stepping in highly dynamic scenes with (a) weakly compressible SPH (2.3M particles), (b) predictive-corrective incompressible SPH (4.5M particles), accelerating computation by 1.8 and 2.0 times respectively over global adaptive time stepping.

Abstract
This paper presents novel and efficient strategies to spatially adapt the amount of computational effort applied based on the local dynamics of a free surface flow, for both classic weakly compressible SPH (WCSPH) and predictive-corrective incompressible SPH (PCISPH). Using a convenient and readily parallelizable block-based approach, different regions of the fluid are assigned different time steps and solved at different rates to minimize computational cost. Our approach for WCSPH scheme extends an asynchronous SPH technique from compressible flow of astrophysical phenomena to the incompressible free surface setting, and further accelerates it by entirely decoupling the time steps of widely spaced particles. Similarly, our approach to PCISPH adjusts the number of iterations of density correction applied to different regions, and asynchronously updates the neighborhood regions used to perform these corrections; this sharply reduces the computational cost of slowly deforming regions while preserving the standard density invariant. We demonstrate our approaches on a number of highly dynamic scenarios, demonstrating that they can typically double the speed of a simulation compared to standard methods while achieving visually consistent results.


Keywords: regional time stepping, asynchronous time integration, SPH, PCISPH

1 Introduction
The Smoothed Particle Hydrodynamics (SPH) method is a powerful and widely used approach to liquid animation [Müller et al. 2003; Monaghan 2005; Becker and Teschner 2007]; among other benefits, it produces detailed splashing and droplet effects, supports seamless topological changes and preservation of liquid mass, and handles complex boundaries in a straightforward manner. However, capturing a sufficiently wide range of spatial scales in order to generate visually compelling results often requires large particle counts, and correspondingly long simulation times.

To date, several acceleration strategies have been proposed to tackle this challenge, including GPU or multi-core CPU methods that exploit parallelism (e.g., [Goswami et al. 2010; Ihmsen et al. 2011]) and spatially adaptive methods that coarsen the particle’s spatial resolution away from the surface (e.g., [Adams et al. 2007; Solenthaler and Gross 2011]). We propose a new and complementary approach.

Across all SPH methods, the choice of time step remains a crucial factor in determining the overall computational cost. All else being equal, the smaller the time step, the more iterations that must be taken to simulate a given span of time, and hence the longer the total time spent running the simulation. The standard time stepping strategy is to use a single global time step which is either held constant throughout the simulation, or varied as a function of the most rapidly deforming region of the flow to ensure stability and accuracy [Ihmsen et al. 2010]. However, many practical fluid flows involve both slow movement and comparatively rapid movement, due to external forces, inflow/outflow boundaries, collisions with objects, and so forth. A global time step is often much too conservative in slow moving regions, leading to a great deal of wasted computational effort where a large time step would suffice.

Problems of this nature suggest the use of asynchronous time integration: different regions of a simulation should be computed at different rates in order to maximize efficiency while satisfying accuracy and stability restrictions. Variations on this idea have been applied to animation problems in rigid bodies, cloth, deformable bodies, and collision processing [Mirtich 2000; Tomaszewski et al. 2008; Harmon et al. 2009], and it has a long history in mechanics (e.g. [Belytschko 1981]). This general strategy has also been developed for certain SPH simulations in astrophysics [Owen et al. 1998; Serna et al. 2003]. However, to our knowledge this concept has not been extended to animating free-surface flow of incompressible fluids...
ible liquids with weakly compressible SPH (WCSPH) [Becker and Teschner 2007], nor has it been applied to the predictive-corrective incompressible SPH (PCISPH) scheme [Solenthaler and Pajarola 2009].

In this paper, we introduce regional time stepping (RTS) approaches for both the WCSPH and PCISPH methods, in which computational effort is expended on different fluid regions in proportion to the speed of their local dynamics. In our numerical experiments, we were able to reduce simulation times by approximately a factor of two compared to global adaptive time stepping on realistic, highly dynamic scenes in which the entire connected body of fluid is in motion. Our algorithm relies on an efficient block-based technique to determine the different regions and support convenient parallelism. After reviewing related work, we will outline how to choose the regions and their corresponding time steps, and then describe how we effectively incorporate this central idea into each of the WCSPH and PCISPH schemes.

2 Related Work

The smoothed particle hydrodynamics method, or SPH, was first applied to liquid animation by Muller et al. [2003], although Desbrun and Cani [1996; 1999] had earlier applied it to animating highly deformable bodies. Further background on the classic weakly compressible SPH scheme can be found in a review by Monaghan [2005] and a paper by Becker and Teschner [2007]. More recently, the predictive-corrective incompressible variant of SPH (PCISPH) introduced by Solenthaler and Pajarola [2009] has been widely adopted because it allows for significantly larger time steps while maintaining incompressibility.

The Navier-Stokes equations governing fluid flow naturally yield behavior spanning a wide range of spatial and temporal scales; depending on the application of interest, certain of these features are more relevant than others. For example, in liquid animation the surface motion and details often take priority, and this is captured in spatially adaptive approaches that coarsen the particle scale further from the surface to reduce the total number of particles [Adams et al. 2007; Zhang et al. 2008; Solenthaler and Gross 2011]. On the other hand, both GPU-based methods [Takahiro Harada 2007; Zhang et al. 2008; Goswami et al. 2010] and multi-core CPU-based methods [Ihmsen et al. 2011] have also been proposed to accelerate SPH simulations, without necessarily relying on adaptivity. The application of asynchronous time stepping is largely orthogonal to many of these approaches, and therefore complementary; we demonstrate our new method within a parallel CPU-based SPH code.

As noted earlier, the time step is a crucial factor in the computational cost of SPH. The most common strategy to incorporate temporal adaptivity is to modify the global time step over the course of the simulation based on the maximum velocities and forces of the entire fluid body at a given time. This allows the simulation to proceed more rapidly during calm motions, but to take much smaller time steps when necessary to resolve very rapid motion. For example, this strategy was recently adapted to the PCISPH method by Ihmsen et al. [2010]. Raveendran et al. [2011] proposed a rather different multi-resolution strategy to allow large time steps: the SPH method is augmented with a broad-scale Eulerian projection method to provide a good initial guess at the fluid pressure. In contrast, our approaches are purely Lagrangian.

We are aware of no methods for liquid animation that exploit the possibility of varying the time step itself spatially. One partial exception is the method of Goswami and Pajarola [2011] in which very slow moving particles are entirely frozen to save computational cost. Although this accelerates the simulation, its applicability is fairly limited and it can introduce objectionable dissipation effects in the fluid motion if applied aggressively.

SPH methods in astrophysics applications have employed asynchronous time integration strategies [Hernquist and Katz 1989; Owen et al. 1998; Serna et al. 2003; Vanaverbeke et al. 2009] to deal with large variations in time scales and stiffnesses. This setting differs from ours in that the target medium is typically compressible and doesn’t involve a free-surface. Our asynchronous approach for weakly compressible SPH builds on that of Serna et al. [2003], augmenting it with a block-based approach that lets us smooth temporal variations between regions and skip a larger amount of computation in less active regions. Furthermore, we develop a novel regional time stepping method for PCISPH that extends many of the advantages of asynchrony to this setting as well.

Lastly, we note that while projection-based Eulerian methods for incompressible flow are inherently synchronous to some degree, Patel et al. [2005] explored using distinct time steps for disjoint liquid bodies of the same simulation to gain some of the benefits of asynchrony.

3 Block-based Computation

Our algorithm relies on a block-based architecture. If \( r \) is the initial particle spacing, we divide the simulation domain into a virtual grid, with each block having support radius \( r \), such that \( r \cdot 2s \). Thus each particle is contained by exactly one of the blocks in the simulation domain.
Such an arrangement has several benefits. For example, neighbors of all particles in a block can be computed efficiently by examining neighboring blocks. Each block can also be treated as a parallelization unit for computing the physics of particles within it, as in the work of Goswami et al. [2010].

However, the most important advantage of the block-based arrangement in our case is parallel region determination. The time steps for a given region are computed over these virtual blocks instead of at the particle level, under the reasonable assumption that liquid in a local area tend to be deforming at comparable rates. This method can then be efficiently parallelized by launching a thread per filled block instead of per particle. Particles falling within that block report their velocity and force up to the parent block, thereby avoiding any race or collision conditions.

### 3.1 Time Step Selection

Our simple block-based time step computation is illustrated in Figure 3, and comprises three steps:

1. All particles compute their velocity and total force.
2. Particles propagate their attributes to their parent (i.e., containing) block. A minimum time step is computed for the block based on the maximum force and velocity from its particles.
3. Each block’s time step is propagated back to its particles.

![Figure 3: Block-based time step computation for particles, red (Δt_n), green (2Δt_n) and blue (3Δt_n) where Δt_n is the base (smallest) time step. (a) Particles colored by the individual time steps they would ordinarily possess. (b) Particles pass their velocity and force values to their parent blocks. Each block is assigned the minimum required time step based on its particles. (c) The block propagates its computed time step back to the particles. (Particles whose time step has been altered are outlined in black.)](image)

This approach is used both for WCSPH and PCISPH.

In what follows, \( \alpha \) denotes a region or set of blocks assigned to a given time step \( \Delta t_n = n\Delta t \beta \) where \( \Delta t \beta \) is the base time step, and \( n \) is a positive integer. The corresponding particle set is denoted by \( S_n \).

A block is assigned to a region corresponding to the largest time step for which it satisfies a set of three criteria, according to its particles’ maximum velocity and force. The first two criteria are:

\[
\Delta t_n \leq \frac{\lambda \theta r}{c_s}
\]  

\[
\Delta t_n \leq \frac{r}{c_{\text{rs}} \ F_{\text{max}}}
\]

These are standard time step conditions from the SPH literature (e.g., [Desbrun and Cani 1999; Becker and Teschner 2007]); the first is a CFL condition, while the second accounts for sudden accelerations over a time step. In these equations, \( c_s \) is the speed of sound in the medium, \( m \) is the particle mass, \( F_{\text{max}} \) is the maximum force magnitude of particles in the block, and \( v_{\text{max}} \) is the maximum velocity magnitude of particles in the block. We set the remaining coefficients \( \lambda \) to \( \alpha \leq 0.4 \) and \( \lambda \leq 0.25 \).

We introduce a third criterion to partition particles into groups depending on their velocities:

\[
\frac{\Delta t_n V_{\text{max}}}{r} \leq \alpha \beta \alpha
\]

In essence, Equation 3 assigns to each particle a time step based on the fraction of its support radius that it would cover in a step moving at its current velocity. The threshold cutoffs \( \beta \alpha \) determine how the time steps are partitioned.

For SPH all three criteria are applied, and we set \( \alpha = 0.4 \). For PCISPH, larger time steps can safely be taken than the classic CFL condition would dictate, so equation 1 is omitted when assigning blocks to regions, and we set \( \alpha = 1 \).

In our implementation, the value of \( \beta \) was set as follows:

\[
\beta_1 = \infty, \ \beta_n = 0.4(0.2)^{(n-2)} \text{ for } n \geq 2
\]

By assigning an arbitrarily large value to \( \beta_1 \) we ensure that \( \beta_1 \) is assigned the smallest time step. The choice of \( \beta_2 \) comes directly from the CFL condition, and higher coefficients are obtained by scaling down the previous value.

### 4 Regional Time Stepping with SPH

Serna et al. [2003] introduced an asynchronous predictor-corrector time integration strategy for their DEVA astrophysical SPH code, later also used by the GRADSPH code [Vanaverbeke et al. 2009]. We begin by briefly reviewing this method, and refer readers to Serna’s work for an expanded exposition.

Given a set of particles assigned different time steps, consider advancing through the union of all the resulting time steps. Beginning from a current time \( t_n \), with positions \( x_i^n \), velocities \( v_i^n \), and accelerations \( a_i^n \) for each particle, the following predictor step of length \( \Delta t = t_{n+1} - t_n \) is taken by all particles to estimate new velocities and positions at time \( t_{n+1} \):

\[
x_i^{n+1} = x_i^n + v_i^n \Delta t + \frac{a_i^n (\Delta t)^2}{2}
\]

\[
v_i^{n+1} = v_i^n + a_i^n \Delta t
\]

Among the set of all particles, the time \( t_{n+1} \) will be the conclusion of a “true” time step for some, called active particles; for the remainder this step is taken only to provide intermediate information to nearby particles. Next, only the active particles have their neighborhoods and accelerations re-evaluated at \( t_{n+1} \), and their positions and velocities are corrected:

\[
x_i^{n+1} = x_i^{n+1} + \frac{(a_i^{n+1} - a_i^n) \Delta t^2}{6}
\]

\[
v_i^{n+1} = v_i^{n+1} + \frac{(a_i^{n+1} - a_i^n) \Delta t}{2}
\]

Crucially, \( \Delta t \) refers to the length of time between \( t_{n+1} \) and the last time that each specific particle’s acceleration was evaluated (i.e.,
the length of its true time step). All other particles maintain their previous acceleration value.

The net effect is that particles taking large time steps assume constant acceleration over their true step, and the intermediate predictor steps approximate the necessary “substep” information required by nearby particles that may be taking smaller (or offset) time steps. Because acceleration is assumed constant (and position and velocity treated accordingly), the number of substeps taken does not change the final end-of-step positions or velocities for the particles being substepped, compared to taking a single large step; the substepping is merely an interpolation process.

The correction applied at the end of a particle’s true time step maintains second order accuracy in position and velocity. Furthermore, without this correction we’ve asynchronous simulations exhibit visual artifacts. As evident from the noisier surface and altered color distribution in the dual dam breaking scenario in Figure 4, the particles’ natural motion and stability is disrupted, erroneously leading to smaller time steps.

4.2 Incorporating Regional Time Steps

Although this approach saves on expensive evaluations of forces and accelerations, it still requires substepping of all particles in the simulation at the smallest global time step. We make the further observation that if all the particles within a given particle’s neighborhood require only the same or larger time step, then no interpolated substeps need to be taken and the final result will be the same. Therefore in regions of our domain assigned large time steps, we can safely integrate all the contained particles at that timestep without the need to perform any substepping whatsoever. This allows the simulation to remain synchronized overall, while correctly integrating different regions at appropriate rates and avoiding unnecessary computation.

The basic outline of our approach is presented in Algorithm 1. The first step assigns a time step to each block within the simulation domain. That is, we choose \( n_m \) and \( S_n \), and update the global block-based neighborhood grid.

To ensure that the time step varies gradually across the physical domain, which aids in simulating quite stiff incompressible flows, we locate the boundary between regions with different time steps, and determine the set of blocks \( \ll n_m \) on the side with the larger time step. This region is then assigned the smaller time step of its neighboring region, which is done efficiently at the block level by checking each block’s neighbors.

In our algorithm, the particles maintain a few additional variables. \textit{validity} is the number of the smallest time steps for which its most recently computed attributes are assumed valid (i.e., how many substeps before its true time step ends). \textit{compute} is a Boolean flag that indicates whether the particle is currently active (i.e., requires recomputation of its acceleration, and end-of-step correction of its position and velocity) which occurs when the \textit{validity} \( \leq 0 \). If the particle is active, its neighborhood set is determined and its local density and forces are computed. Otherwise, it skips these steps. At the end of each loop, the position and velocity of each particle is updated, and active particles have their velocities and positions corrected (lines 25-30), per Serna’s scheme [Serna et al. 2003].

We make some additional observations. First, while the computation of time steps is determined per block, it is updated on the individual particles which also track their own validity. Blocks do not have validity or history, and therefore all computations over blocks are valid only for a frame. Second, the algorithm is pre-emptive. That is, a particle can change its time step even before its validity expires (line 14 of Algorithm 1). This allows the method to maintain stability in the face of sudden accelerations, as often occurs in collisions with boundaries. Figure 5 illustrates a double dam break simulated using this scheme.

5 Regional Time Stepping with PCISPH

5.1 Motivation

The second major contribution of our work is to develop a regional time stepping method for predictive-corrective incompressible SPH [Solenthaler and Pajarola 2009]. As the name suggests, PCISPH enforces incompressibility through a predictor-corrector approach, which iteratively refines pressure forces to correct any deviations in particle density. This allows for time steps about an order of magnitude larger than weakly compressible SPH, while recovering near-identical behavior. To accelerate this method, IhmSEN et al. [2010] proposed an adaptive time stepping PCISPH scheme that adjusts the global time step depending on the simulation state. This does indeed improve the speed of PCISPH, however its overall efficiency is limited by fast-moving regions, which can arise frequently during collisions with boundaries.

Similar to our method for WcSPH, our essential observation is that slow moving regions should require less computational effort to simulate a given amount of time. Concretely, for PCISPH, this is because the particle density changes more slowly in these regions, and therefore these density variations ought to require fewer corrective iterations to resolve. (In their adaptive time stepping work, IhmSEN et al. noted the converse: fast motion and large impacts can require many more density correction iterations.)

The second observation we build on is that more localized density corrections can be highly effective. Raveendran et al. [2011] noticed this, and exploited it by applying a post-process that spends extra iterations correcting only those particles with large remaining density errors after their core algorithm concludes. We instead make this observation a fundamental feature of our algorithm, lo-
Algorithm 1: Regional Time Stepping for Standard SPH

1: for all particles \(i\) do
2:     set \textit{validity}_i = 0
3: while (animating) do
4:     update block neighborhood grid
5:     /*——— Region determination \(<\textit{S}_i\) ———*/
6:     for all \(\in \textit{S}_i\) do
7:         update parent block maximum with \(\mathbf{v}_i\) and \(F_{i\text{total}}\)
8:     for all blocks \(b\) do
9:         compute new region membership per section 3
10:    for all \(\in \textit{S}_i\) do
11:        decrement \textit{validity}_i
12:        if (\textit{validity}_i \leq 0) \&\& (parent block has a different time step) then
13:            set \textit{compute}_i = true
14:        else
15:            compute \textit{validity}_i, timestep
16:        end
17:    /*——— Physics computation ———*/
18:    for all \(\in \textit{S}_i\) do
19:        if \textit{compute}_i then
20:            find neighborhoods \(\mathbf{N}_i\)
21:    for all \(\in \textit{S}_i\) do
22:        if \textit{compute}_i then
23:            update \(\rho_i, p_i\)
24:    for all \(\in \textit{S}_i\) do
25:        if \textit{compute}_i then
26:            predict new \(\mathbf{x}_i\)
27:        apply correction to \(\mathbf{v}_i\)
28:        apply correction to \(\mathbf{x}_i\)

Figure 5: A 1.5M particle double dam break simulated using our regional time stepping algorithm for WCSPH. Top: Particles colored by time step region. Bottom: Particles for corresponding frames from a different viewpoint and colored uniformly.

cally applying a different number of density correction iterations based on the dynamics of different regions of the flow.

5.2 The Algorithm

At a high level, our algorithm works as follows: we pick a large time step \(t_n\), called the major time step, and divide it into \(n\) equal subintervals, or minor steps, so that \(t_n = n\Delta t_b\). (We used \(n = 4\).) At the beginning of a major step, we assign blocks of particles to different regions based on their dynamics as in section 3. On each minor step, all regions perform at least one iteration of density correction, and then participate in additional iterations depending on their region membership.

Note that our algorithm is therefore not truly asynchronous in the manner of our RTS SPH approach; all particles are advanced in synchronization. However, the computational expense of slow moving regions is dramatically reduced, by lowering the number of correction iterations applied. On the other hand, we do update the particle neighborhoods asynchronously in proportion to how fast they are likely to change, since neighborhood searches are a major expense in SPH algorithms. As in WCSPH, we maintain \textit{validity} variable for each particle that tracks how many (minor) time steps a particle’s data is considered to be valid for, based on its region membership. Primarily, this means that we update the particle neighborhood only when its validity expires, and otherwise reuse its most recently computed neighborhood.

Pseudocode for our approach is given in Algorithm 2, and we describe its various elements below.

5.3 Global Density Correction Schedule

On each minor time step, we perform a certain number of iterations of density correction which varies by region. The number of iterations applied to each region type is determined by the density correction schedule shown in Figure 7. This schedule which was chosen heuristically to satisfy certain constraints. Specifically, a minimum of one correction iteration should be applied to all particles to keep them minimally synchronized. Within the “true” time step for a given particle it should cumulatively receive at least 3 correction iterations (similar to standard PCISPH). Over a given major
Block-based region determination.

(a) Blocks are assigned time steps.
(b) At borders between time step regions, we assign blocks with larger time steps (blue, $3t$) the time step of their neighbour with a smaller time step (green, $2t$) to smooth out region transitions. This region is called $\mathcal{R}_{\text{min}}$, shown with brown borders.
(c) For RTS PCISPH, we must also identify the set of blocks $\mathcal{R}_{\text{ob}}$ which border region changes on the side with the larger time step; this region is observed for density variations of particles that were previously resolved.
(d) We also identify the region $\mathcal{R}_{e}$ containing particles with remaining density errors regardless of their region, and augment $\mathcal{R}_{\text{ob}}$ with the cells bordering this region.

Our density correction schedule for PCISPH. Each column (a)-(d) represents one minor time step within a major time step of $4t$. Each row within a column represents one iteration of density correction being applied. The colored blocks within each row indicate which regions have density correction performed on them at a given iteration within a minor step. Blocks labelled $e$ correspond to iterations performed for all remaining particles whose density error hasn’t been fully corrected.

The schedule we designed has the following properties. All regions receive a minimum of one iteration of density correction on every step, as seen in the first row, with the exception of the very first step within a major step, in which all regions receive two iterations. Additional iterations are then assigned based on region membership.

The fastest moving ($t$) regions receive 3 iterations per step, as in classic PCISPH. The next fastest moving ($2t$) region receives 3 iterations for every two minor steps. Slower moving regions similarly receive fewer total iterations per major step.

Figure 6: Block-based region determination. (a) Blocks are assigned time steps. (b) At borders between time step regions, we assign blocks with larger time steps (blue, $3t$) to smooth out region transitions. This region is called $\mathcal{R}_{\text{min}}$, shown with brown borders. (c) For RTS PCISPH, we must also identify the set of blocks $\mathcal{R}_{\text{ob}}$ which border region changes on the side with the larger time step; this region is observed for density variations of particles that were previously resolved. (d) We also identify the region $\mathcal{R}_{e}$ containing particles with remaining density errors regardless of their region, and augment $\mathcal{R}_{\text{ob}}$ with the cells bordering this region.

5.4 Local Density Correction

We sometimes find that a small number of particles have unresolved density errors after their assigned number of correction iterations is completed on a given step. We address this in a manner similar to Raveendran et al. [2011].

We introduce an additional region $\mathcal{R}_{e}$ (with particle set $S_{e}$); this constitutes all particles having error larger than a chosen threshold irrespective of their time step. We follow the same error metric as given by Ihmsen et al. [2010] where the average density error $\rho_{\text{avg}}$ of all particles and the maximum density error $\rho_{\text{max}}$ of any particle should not exceed a certain value. Per Raveendran et al., this threshold is set to half the value of the largest allowed density error. The particles that compose $S_{e}$ are corrected at every iteration of our schedule. If no such particles exist, we skip this step.

Since we often perform density correction on just a subset of all the particles, we need to know if these corrections have disrupted the density of particles whose density was previously declared correct. Movement near or across the boundaries between regions can easily cause this to occur. To account for this we identify another region, $\mathcal{R}_{\text{ob}}$, which is the set of “observed” blocks that we monitor for changes, as shown in Figure 6. This is the set of blocks that border on either $\mathcal{R}_{e}$ or a region with a smaller time step; its particle set is denoted $S_{\text{ob}}$. After each iteration, it suffices to check just the density of particles in $S_{\text{ob}}$ for newly introduced errors, avoiding an expensive global check on all particles whose density is already correct.
5.5 Extra Correction Iterations

After the standard 3 iteration schedule is completed for a given minor step, if some error in density remains we do one of two things. We can either perform additional purely local corrections on the particles with remaining error, or run through additional global iterations by resetting to the top of the correction schedule for the current minor step. Local correction is often preferable since it minimizes the number of particles involved, but it may not always be efficient or feasible if the error is large or global. We therefore make this choice by considering whether the average compression error $\rho^{avg}$ exceeds a threshold, since this is indicative of global density errors. If local iterations are initially selected, but they nonetheless fail to converge after a few iterations, we switch back to global corrections and revert the pressure estimates to their values from before local iterations began.

If a minor step exceeds 6 global iterations for correction, we terminate the major step early and temporarily reduce the length of subsequent major steps to $2\Delta t$ (i.e., two minor steps rather than four). This allows the global neighborhood grid to be completely updated more frequently to better handle these large density changes. After 10 major steps in which we do not exceed 6 global iterations on any minor steps, the major time step is reverted. However, we emphasize that in our experiments we found that for the majority of cases extra global operations aren’t necessary; a few steps of local correction typically suffice.

For PCISPH we make one additional minor change to our region determination strategy: we expand regions $< 1$ and $< 2$ by 4 layers of blocks each before beginning a major step, in a manner similar to $Re_{min}$. This is done to account for fast particles that may travel several blocks from their original position.

5.6 Neighborhood Computation and Updates

At the start of each major step, we compute the block grid and the particles contained in each block. Particles then compute their neighborhood using this grid, along with their region membership and the number of minor steps for which they are considered valid. This neighborhood will be used to compute external forces and the pressures needed to correct density variations. Since block updates and neighborhood search are generally expensive operations, we prefer to avoid them as much as possible, through the use of an asynchronous neighborhood update strategy. Particles assigned to faster regions update their neighborhoods whenever their validity expires, at the start of a minor step. That is, particles in region $< 2$ update their neighborhood every step, particles in $< 2$ update every second minor step, and particles in $< 4$ do so only once per major time step. We found that since particles in $< 3$ are also slow moving, postponing their neighborhood update to the next major step is satisfactory as well.

At each minor step, we also check if a particle now requires a smaller time step. If so, we mark its block and the neighboring blocks to update their time step. This allows the simulation to adapt to sudden changes.

When a particle’s validity expires and its neighborhood needs updating, a naïve approach would globally update the particles belonging to each grid block and performing a new search within the updated neighboring blocks. This would be very costly, limiting the benefit of our asynchronous strategy. To cope with this, we take two steps. First, we initially over-populate the grid at the start of each major step, then we use a grid with blocks of size $r = 2s$ rather than $r = 2s$. This ensures that particles will nearly always find all the particles belonging to their neighborhood in their own or adjacent blocks, even if we keep the same block data over the entire major step.

Second, for the very fast particles within $< 1$, we expand the radius of our neighbor particle search to two grid blocks instead of one (see Figure 8). While this means we must check many more possible neighbor particles, this is justified because these particles are few in number and arise infrequently, yet are critical to stability. Doing this with the particles in $< 2$ as well would be substantially more expensive. Figure 10 illustrates that expanding the search radius for both $< 1$ and $< 2$ makes little visual difference, so we avoid doing so for the sake of efficiency. (Moreover, standard PCISPH assumes a fixed neighborhood over large time steps [Solenthaler and Pajarola 2009], so the method is reasonably robust to small errors in neighborhood estimates.)

6 Results

The proposed and baseline WCSPH/PCISPH methods were implemented on a MAC 10.7.5 machine with 3.2 GHz quad-core Intel processor, using C++ and the OpenMP API. The images were rendered offline with POVRAY.

For WCSPH, we have used $\Delta t_{gfl}$ for standard time stepping and $\Delta t_B = \Delta t_{gfl}$ for RTS. For PCISPH, we compare global time stepping with a constant time step against both adaptive global stepping [Ihmsen et al. 2010] and our method. For the sake of experimental similarity with Ihmsen’s results, we have used a constant time step of $0.000166$ for standard PCISPH method and no static boundary particles. For both global time stepping PCISPH and RTS PCISPH we use static boundary particles. Inactive static boundary particles are culled prior to computing the physics.

For the scenes in Table 1, $\Delta t_B$ is chosen to be $0.00035$ or Figure 2 and 9, and $0.0003$ for Figure 12. This essentially allows a larger time step of $4\Delta t_B$ in each major cycle. The user-chosen error threshold $\rho_T$ (for whether to perform extra global or local corrections) is set to 0.25 in all our experiments.

Since we a moderately restrictive value of $\alpha = 0.4$ in Equation 3 for RTS WCSPH, the region variation is more sensitive, particularly between regions $< 2$ and $< 3$. For the same reason, particles do not often take time steps larger than $3\Delta t_B$. Our block-based RTS algorithm allows these transitions regions and occasional very fast moving particles to be treated efficiently, while keeping the simulation stable and accurate.

Table 1 gives the performance speed-up of our methods in comparison to both WCSPH and PCISPH for several examples. Our methods yield simulations between 1.7 and 2.1 times faster than the comparison methods for the given examples. Table 2 provides relative timing breakdowns for our methods into physics compu-
Algorithm 2: Regional Time Stepping for PCISPH

1: while animating do
2: update block neighborhood grid
3: /*——— Region determination (\( \leq i \)) ———*/
4: for all \( i \in S \) do
5: update parent block maximum with \( v_i \) and \( F_i^{\text{total}} \)
6: for all blocks \( b \) do
7: compute new region membership
8: expand regions \( \leq 1, \leq 2 \)
9: find observed blocks \( R_{ob} \) and particle \( S_{ob} \)
10: for all \( i \in S \) do
11: update timestep \( t_i \), validity \( i \), compute \( i \) using parent block
12: /*——— N Minor time steps ———*/
13: \( j = 1 \) to \( N \) do
14: for all \( i \in S \) do
15: \( \text{if compute } i \) do update neighborhood \( N_i \)
16: for all \( i \in S \) do
17: \( \text{if compute } i \) do update \( F_{i}^{\text{proj}} \)
18: for all \( i \in S \) do
19: initialize pressure \( p(t) = 0.0 \)
20: initialize pressure force \( F_{i}^{\text{proj}} = 0.0 \)
21: /*——— Density Correction ———*/
22: DensityCorrectionRTS
23: for all \( i \in S \) do
24: compute new \( v_{i}(t + 1) \)
25: compute new \( x_{i}(t + 1) \)
26: for all \( i \in S \) do
27: decrement validity \( i \)
28: if (validity \( i \leq 0 \)) then
29: set compute \( i = \text{true} \)
30: update validity \( i \)
31: else
32: compute \( i = \text{false} \)

3: while \( \text{iter} \leq \text{minIterations} \) do
4: for all \( i \in S \) do
5: predict velocity \( v_{i}^{\star}(t + 1) \)
6: predict velocity \( x_{i}^{\star}(t + 1) \)
7: for all \( i \in S \) do
8: if \( (i \text{ has turn) } k (i \in S_{a}) k(i \in S_{ob}) \) then
9: update density \( \rho_{i}^{\star}(t + 1) \)
10: update density \( \rho_{i}^{\star}(t + 1) \)
11: update pressure \( p(t) = f(\rho_{i}^{\star}(t + 1)) \)
12: if (density error remains outside active regions) then
13: update \( S_{e}, S_{ob} \)
14: for all \( i \in S \) do
15: if \( (i \text{ has turn) } k(i \in S_{e}) \) then
16: compute pressure force \( F_{i}^{\text{proj}}(t + 1) \)
17: \( \text{iter} ++ \)
18: if \( (\rho_{i}^{\star}(t + 1) \geq \eta) \& (\text{iter} \geq 3) \) then
19: minIterations ++
20: if \( (\rho_{i}^{\star}(t + 1) \geq \eta) \) then
21: reset to top of active step’s correction schedule (i.e.,
22: extra global corrections)
23: else
24: perform local correction on \( S_{e} \)

7 Conclusions and Future Work

We have presented an efficient technique for regional time-stepping for WCSPH and PCISPH. The proposed methods are simple to implement and can achieve significant speed-ups while maintaining behaviour consistent with the corresponding fully synchronous simulations. We envision several directions for future work. Two natural extensions would be implementing a GPU-based version, and combining our method with spatial adaptivity, such as Solenthaler’s...
two-scale method [Solenthaler and Gross 2011]. It could also be useful to adjust the base step of our PCISPH scheme in a manner similar to the adaptive global time stepping method of Ihmsen et al. [2010], in order to achieve the benefits of both. Finally, to reap the full benefits of asynchrony in a wider range of scenarios we would like to explore methods that can couple between SPH and other asynchronously integrated physical systems, such as deformable or rigid bodies.

Table 2: Timing breakdowns for each method. Computations specific to RTS constitute only 10-15% of the run-time.

<table>
<thead>
<tr>
<th>Method</th>
<th>$T_{\text{neigh}}$</th>
<th>$T_{\text{physics}}$</th>
<th>$T_{\text{RTS}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCSPH</td>
<td>37%</td>
<td>42%</td>
<td>14%</td>
</tr>
<tr>
<td>PCISPH</td>
<td>28%</td>
<td>56%</td>
<td>12%</td>
</tr>
</tbody>
</table>

References


Figure 9: RTS PCISPH compared with standard PCISPH. Our method produces overall behavior and features close to the original. (a) RTS particles colored according to their time steps, (b) RTS particles uniformly colored for comparison purpose, (c) RTS surface for corresponding frames, (d) standard particles and (e) standard surface for corresponding frames.

Figure 10: Extending the neighborhood search radius to two blocks for particles in both $<1$ and $<2$ results in little evident difference when compared to 9(a), in which only the search radius for $<1$ is expanded.

Figure 11: A dam break scenario where immediately after releasing the dam we drop a second block of liquid on top, simulated with 2.2M particles using RTS WCSPH.

Figure 12: A highly turbulent flow involving several obstacles and boundaries, simulated with 4.5M particles using the proposed regional time stepping algorithm for PCISPH.