International Journal of Research in Engineering, Science and Management
Volume-3, Issue-6, June-2020
www.ijresm.com | ISSN (Online): 2581-5792

A Comparative Performance Study of Grid-based and Particle-based Fluid Simulation Algorithms

Tushar Singh[1*], Nalin Pandohi[2]
[1,2]*Student, Department of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India*
*Corresponding author: tusharone21@gmail.com

***Abstract*: Simulation of fluids is quite a challenging task because unlike solid objects liquid shows a drastic amount of change per frame hence rendering liquids becomes quite a computationally expensive task. Over the years many new and improved fluid simulation algorithms have been introduced but most of them are fairly complex and put a heavy computational load on the CPU and GPU. In this paper we are going to examine the two most popular fluid simulation algorithms, the lagragian based particle approach and the euler based grid approach. These algorithms are particularly popular because of their simplicity and low computational loads; we will be testing these algorithms by providing real-time interactive workloads to compare their performance.**

***Keywords*: Euler model, Fluid simulation, Gaussian kernel, Grid fluids, Lagragian model, Smooth particle hydrodynamics, Staggered grids, Volumetric Raycasting.**

1. Introduction

Animated fluids are used for a variety of applications which include games, machine modeling and fluid mechanics. Fluids are one of the highly disoriented states of matter because their particles move about in a haphazard motion and colliding with everything around them whether it be a liquid or gas. Animation of such fluids are very difficult because of the nature of these states of matter, it is also very difficult to animate their interactions with solid objects like walls, boundaries etc because one must accurately depict how the liquid will behave under such conditions. Although the liquid surface seems like a continuous film but instead the liquid consists of several small particles moving about in motion, the entire behavior of the liquid is determined by the motion and interaction of these small particles.

Fluid simulation use actual hydrodynamics equations to animate the movement of fluid particles, since they are many ways to model fluid particles there are many algorithms to simulate fluids but in our study we are focusing on grid based euler approach where the fluid particles were monitored and updated as they passed through certain grid cells and the particle based lagragian approach where discrete blobs of fluids particles are used for simulation and tracking. Over the years there have many new developments in these algorithms. Chen [1] proposed an improved SPH solver that included vorticity confinement to capture the small scale details.

The motion of the fluid which is usually not captured by the standard SPH algorithms. Shao [2] suggested a smoothing constant to be added to the SPH vorticity equation that help reduce the unstable boundary problems. Li [3] suggested some

kernel functions and field quantity equations that could help with surface rendering. Chen [4] proposed a nearest neighbor octree searching method that reduces the time required by the algorithm to find and compute pressure forces from close particles. Mould [5] suggested to use a regular grid on top of tall cells that would help render the volume below the liquid surface without using extra compute power. Chentanez [6] suggested to use a grid simulation technique where the grid cells would be deformed to help in the boundary transitions of the fluid. Kipfer [7] suggested a technique to remove unused cells from a grid to help lower the computations performed per second. We will be incorporating all these works into our algorithm so that we can compare the most updated techniques and find out areas of improvement and possible areas of application for both the techniques.

A. Governing Equations

The base governing equations for both the algorithms is the Navier stokes equation normally used in fluid particle hydrodynamics. In this study however we are neglecting some of the highly complex features like turbulence, surface tension etc. Also since we are simulating a single fluid ie water we are considering the fluid to be incompressible which helps save a lot of compute power and has a little effect on the visual appearance of the fluid.

The grid based and particle -based approaches are based on the Navier stokes fluid mechanics equations, but since we are dealing with animated fluids therefore we can remove some aspects like slightly compressible and treat them like incompressible and also viscosity is removed from the Navier stokes equations because of the single fluid nature of the test set. The resulting Navier stokes equations are-

$$\frac{\partial \vec{u}}{\partial t} + \frac{1}{\rho}\nabla p = \vec{F} \quad \text{and} \quad \nabla . \vec{u} = 0 \tag{1}$$

These equations denote the conservation of momentum and mass, where F represents the external forces like gravity, ρ is the density and $\vec{u}$ is fluid velocity while p being the pressure inside the fluid. These reduced equations are used in this algorithm to predict the movement of fluid particles. As far as the notations are concerned we will be using standard physics notations to represent the equations and all symbols are properly defined after an equation is introduced.

2. Fluid Simulation Algorithms

In this section we will discuss in detail the steps involved in both the algorithms and also indicate the modifications done by including relevant algorithms from previous papers.

In abstract terms, we can define the grid based and particle based algorithms with an example- suppose if we want to capture the movement of a bird in a park, we could use two approaches to do that firstly we can set up a camera at a specific point in a park and capture the birds movement but we will be unable to track it once it moves outside the camera view. This is the grid approach we define certain grid cells where the moving particles are monitored and then updated in terms of their properties. Secondly we could follow the bird around this forms the particle basis of particle based approach, we are looking at the simulation from a particle in a fluid flow point of view. This method is simpler and faster because no extra computations need to be performed to account for loss of information.

A. Grid-based Algorithms

The grid based algorithms sample the scalar and vector fields at various fixed points in space and interpolates the values in between these points. Now we will discuss the implementation details of these algorithms.

1) Data Structures

The grid based approach stores the data of different particles namely velocity, pressure, direction etc for this they use an approach called the staggered MAC grid which is suggested by Mould[5], this grid is quite different from a single grid as such that the pressure is stored the very centre of the grid cells but the velocity of the fluid particles is stored on the sides, these velocities are the normal component of the particle velocity passing through the cell. They use this data structure because since the grids monitor the particles at specific positions if they want to find velocity at point i between two points in the grids then we can calculate the derivative in such cases by using the two grid point values but the actual value at I is ignored but with the staggered grid we can find the derivative without loss of information.

$$\frac{\partial w}{\partial x_i} \approx \frac{w_{i+\frac{1}{2}} - w_{i-\frac{1}{2}}}{2\Delta x} \tag{2}$$

Where w is a any fluid property sampled at different locations in the simulations space $w_0 \ldots . w_n$, i denotes the point where we need the central difference. We use central differences because to find the gradient we must find the change in property and the two sides of the MAC cell stores these values. As we can see the formulae uses half indices but in a practical sense these indices will not work they only exist in a theoretical sense therefore, equations must be defined to map these to real values.

$$p[i][j][k] = p_{i,j,k} \tag{3}$$
$$u_x[i][j][k] = u_{i-\frac{1}{2},j,k} \tag{4}$$
$$u_y[i][j][k] = u_{i,j-\frac{1}{2},k} \tag{5}$$

Where u represents the velocity of the fluid particle and i, j, k are the components in three dimensions. These equations help us map half indices to actual values in the staggered grids.

2) Time-step

Now we will try and explain the steps involved in the algorithm, first an accurate timestep must be chosen the timestep tells that at what intervals must the particle properties be recalculated and updated, this must be chosen as such so that the particle is updated when it reaches the next grid point without skipping any cells in between.

$$\Delta t = \frac{\Delta h}{\vec{u}_{max}} \tag{6}$$

$$\Delta t = k_{C\,F\,L} \frac{\Delta h}{\vec{u}_{max}} \tag{7}$$

$$f(v, v_{max}, v_{min}) = \kappa_{min}(v - v_{min}) \frac{\kappa_{max} - \kappa_{min}}{v_{max} - v_{min}} \tag{8}$$

$$\vec{u}_{max} = \max(|\vec{u}|) + \sqrt{\Delta h |\overrightarrow{F}|} \tag{9}$$

Where Δt is the time step, Δh the distance moved by the particle during the time step. The $k_{C\,F\,L}$ is the factor that helps satisfy eqn(6) while choosing a time step of your choice. Equations (8) and (9) help us calculate the velocity max and the time step factor by plugging in the respective values.

3) Advection

Next up is the process of advection, this is the process of determining the value of nay quantity let it be pressure, velocity at a later time. Let Q be any measurable quantity of a fluid then to predict it at later time Δt, we use backwards trace method but it requires an extra copy of the properties be stored in each cell. The following equations are used to predict the quantity Q at any later time where Q^n and Q^{n+1} is the quantity after time step Δt.So an overall fluid flowing through the pipe etc will be rendered using the advection rules given below and for the boundary conditions we simply clamp the grid coordinates hence the algorithm does not perform well with boundaries present.

$$Q^{n+1} = advect\left(Q^n, \Delta t, \frac{\partial Q^n}{\partial t}\right) \quad (10)$$

The new value is simply computed by finding the gradient of the previous values and using the timestep in all the three dimensions and then simply update the particle properties as such.

4) Calculating Pressure

Now that we defined a way to predict the properties of the particles at different times, now we must take care of the pressure and incompressibility conditions. The particles moving around in the grid space must not cause the overall volume to increase because as indicated earlier we have considered the fluid to be incompressible. Therefore, in the next step we try to maintain the constant volume by defining the pressure relations among the different particles moving around the grid points. As per physics the incompressibility relations are-

$$\nabla\,(\vec{u})^{n+1} = 0 \quad and \quad (\vec{u})^{n+1}.\hat{n} = \overrightarrow{u_{solid}}.\hat{n} \quad (11)$$

These two equations define the boundary conditions as well, now as per our advection equation we must update the velocity of the fluid particles at after time Δt, here we see that we have no information about any cell that does not any fluid therefore we must assume that the pressure in those cells is zero, now using this information we can compute an equation to update the pressure in very grid cell. The following equations for 2D grid can be defined by using the incompressibility condition.

$$\vec{u}^{n+1}_{i+\frac{1}{2},j} = \vec{u}^{n}_{i+\frac{1}{2},j} - \Delta t\,\frac{1}{\rho}\frac{p_{i+1,j}-p_{i,j}}{\Delta h} \quad (12)$$

$$\vec{v}^{n+1}_{i,j+\frac{1}{2}} = \vec{v}^{n}_{i,j+\frac{1}{2}} - \Delta t\,\frac{1}{\rho}\frac{p_{i,j+1}-p_{i,j}}{\Delta h} \quad (13)$$

And for 3D the equations can be defined as –

$$\vec{u}^{n+1}_{i+\frac{1}{2},j,k} = \vec{u}^{n}_{i+\frac{1}{2},j,k} - \Delta t\,\frac{1}{\rho}\frac{p_{i+1,j,k}-p_{i,j,k}}{\Delta h} \quad (14)$$

$$\vec{v}^{n+1}_{i,j+\frac{1}{2},k} = \vec{v}^{n}_{i,j+\frac{1}{2},k} - \Delta t\,\frac{1}{\rho}\frac{p_{i,j+1,k}-p_{i,j,k}}{\Delta h} \quad (15)$$

$$\vec{w}^{n+1}_{i,j,k+\frac{1}{2}} = \vec{w}^{n}_{i,j,k+\frac{1}{2}} - \Delta t\,\frac{1}{\rho}\frac{p_{i,j,k+1}-p_{i,j,k}}{\Delta h} \quad (16)$$

Where the p indicates the pressure at a certain point in the grid and the rest of the symbols indicate the usual meanings with u, v, w be the three components of velocity in three dimensions.

At the same time we also must take care of the boundary conditions and realize what should happen if the particle ever hits the boundary, for this process we will Dirichlet's condition and Neumann conditions to realize inter particle collision and collision with a solid wall.

Here also we use the compressibility equation to bound the particles within the grid and by making sure that the spatial derivatives are zero, we can approximate the divergences of the updated properties to satisfy the condition, hence we can use the difference equation eqn (2) and update it to come up with a approximation formulae.

$$(\nabla.\vec{u})_{i,j,k} \approx \frac{\vec{u}_{i+\frac{1}{2},j,k} - \vec{u}_{i-\frac{1}{2},j,k}}{\Delta h} + \frac{\vec{v}_{i,j+\frac{1}{2},k} - \vec{v}_{i,j-\frac{1}{2},k}}{\Delta h} + \frac{\vec{w}_{i,j,k+\frac{1}{2}} - \vec{w}_{i,j,k-\frac{1}{2}}}{\Delta h} \quad (17)$$

Now we have all the values required to update the properties of our fluid particles and handle the boundary transitions that might occur. But we must also come up with an equation that combines the formulas listed and come up with a unified equation to calculate the pressure in each cell by keeping in mind the boundary condition as well.

We use a linear equation for the new pressure in each grid cell and then combine all the individual grid equations to come up with a system of simultaneous linear equations that we can solve for the entire grid.

$$\begin{bmatrix} -\Omega_1 & \beta_{1,2} & \cdots & \beta_{1,n} \\ \beta_{2,1} & -\Omega_2 & & \vdots \\ \vdots & & -\ddots & \beta_{n-1,n} \\ \beta_{n,1} & \cdots & \beta_{n,n-1} & -\Omega_n \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{n-1} \\ p_n \end{bmatrix} = \begin{bmatrix} -D_1 \\ -D_2 \\ \vdots \\ -D_{n-1} \\ -D_n \end{bmatrix} \quad (18)$$

Where D_i is the divergence through cell I and Ω_i is the no of surrounding particles around the point considered in the grid cell. β_{ij} is 1 if i and j are adjacent and 0 otherwise. Now we can update the particle properties of all our cells without any difficulties but we must also find a way to track the liquid surface so that the flow can be modeled.

5) Surface Tracking

Finally we must decide how to track the surface of the liquid, this is done via marker particles now suppose we have a grid containing fluid particles each of the grid cells which contains fluid particles is marked as fluid and it's properties are updated as such and the rest are solid and boundary and as the liquid flows we change the marker status of the cells based on the flow direction.

Next to plot the surface of the liquid ,we must find all the grid cells in the 3D space which contain the fluid this is achieved by defining a new distance metric, $\Phi_{i,j,k}$ which is stored at the centre of each cell to compute it we use the position vector $\vec{X}$ and the minimum distance from the set of all points in the grid set., to check whether a point is inside the fluid or not we simply check the value of distance metric Φ, if it less than 0 then inside liquid if greater than zero then outside liquid. Also we continuously advect these values along with other properties like velocity etc and model the flow using the same.

$$D_S(\vec{X}) = min_{\vec{p}\epsilon S}\|\vec{X} - \vec{p}\| \quad (19)$$

The formula above can used for computing signed distance, this also tells us whether a grid cell is inside a fluid or not .The signed distance can used for computing the metric Φ, we have based the signed distance calculation based on Howes [8] algorithm that can track the fluid area very efficiently. Also after tracking the liquid surface we can remove grid cells [7] that do not contain the fluid particles or in other words do not compute pressure of cells where there is no liquid, a simple approach that saves a ton of compute power.

B. Particle-based Algorithms

Now we will discuss the implementation details of the particle based approach. First we decide what all properties must be stored along with each of the particle, since we have to make sure that all the details which are required by the Navier stokes equation must be stored along with particles therefore we stored the position $\vec{X}$, velocity $\vec{V}$, Mass M, density d, pressure p and force $\vec{F}$, now we must decide on the data structure to be used since new particles will have to be added very now and then therefore to store the particles properties a linked list has been used. Now that we have decided on the properties we now define the equations the particles must follow throughout the simulation, for all particles P_i –

$$\frac{\partial \vec{V}}{\partial t} = \vec{A}_i^{pressure} + \vec{A}_i^{viscosity} + \vec{A}_i^{gravity} + \vec{A}_i^{external} \quad (20)$$

The above equation is simply an extension of the Navier stokes equation and it defines the acceleration on the particle due to any of the forces like pressure etc will result in change of the velocity, as mentioned in Navier stokes second equation and Newton's second law.

1) Calculating Pressure

Now we must find out how to calculate the particle properties like pressure after a certain time, properties like mass are not expected to change but pressure must be calculated after every timestep.

As we have a lot of particles in the simulation therefore to calculate the pressure at a certain point we must use discrete summation which is done via the following equation-

$$\sum_{j\neq i}^{n} M_j W_{R_{ij}} \quad (21)$$

Here R_{ij} is the Euclidean distance between the particle i and j, the function W(d) is known as the kernel function as discussed earlier. This function takes in a scalar parameter which in this case is the Euclidean distance between the particles and then returns another scalar which is between 0 and 1. Now this kernel function maps particles which are further away from the point of consideration to values closer to zero which signifies that particles which are far way from the point of consideration do not have any effect on the pressure at that point. Deciding on which kernel function is better is a topic of research but in our project we have used the Gaussian kernel function which takes in a smoothing width h as suggested by Li [3], so particles further than 2h are ignored during the pressure calculation.

$$W(d) = \frac{1}{\pi^{\frac{3}{2}}h^3}\exp(\frac{r^2}{h^2}) \quad (22)$$

Where W(d) denotes the kernel function, r is the effective distance of the blob and h is the smoothing width.

Therefore, for each of the particles we must get list of neighbors, calculate the density for that particle while considering the neighbors and do the same for pressure. Then we calculate the acceleration for that particle using density and pressure and actually move the particle in the simulation as per the calculated acceleration per timestep. The process is repeated until the simulation ends.

2) Acceleration Structures

As we saw that we must find all the neighbors for each of the particles every time we update the particle properties, hence we must add spatial data structures that can help reduce our time complexity because as per the process described above calculation interaction forces between every particle will take $O(n^2)$ time which is not viable since we are trying to reduce the overall computational expenses as compared to the grid based algorithm.

Now we will see how these spatial data structures work, in addition to storing our particle properties in a linked list we also store them in a spatial grid structure or a an octree[4], now this is not like the grid based algorithm we are only storing the particle properties in a spatial grid to facilitate access and reduce I/O overhead. The grid cells form a network over the simulation and each of them extend by a distance of R in all dimensions. This in turn helps us to examine only grid cells for 2D representations and 27 grid cells in case of 3D representations in order to calculate the forces on any particle.. The rest of the grid cells which are farther away are totally ignored because of their less influence on the current particle. Since the particles are in constant motion therefore we must continuously add and remove them to grid cells as they move in or out of them. With this we can now construct a 2D implementation our SPH particle based algorithm with discrete blobs, now since the computations done while finding the forces on a particle due to other particles are independent of one another, ie. we can compute the forces between particle A & B and we can compute the forces between A and C separately on different CPU/GPU cores since they do not exchange information therefore this algorithm is better suited for modern day machines which exploit the parallelism of programs to speed up the execution process.

3) Surface Tracking

Up until now we have provided a way to render discrete blobs of particles and defined how they will interact with one another but when we talk about fluid simulation we do not expect to see

water blobs moving around in a glass but instead we hope to see a smooth fluid surface, though we were able to reproduce the atomic interactions in a fluid but now we must render them from a human's perspective.

Since we have already have a grid structure that contains the properties of particles at different points, we can simply feed these grids to an iso surface volume rendering application and we can get a 3D ray traced output, now in this project we have used direct 3D rendering done via volume raycasting since it is faster but there are other methods that are more accurate. The algorithm can be understood as Suppose we have a volume of any shape with density D(x,y,z) which is penetrated by a single ray R from a distant light source. At each point along the ray entering the volume we have an illumination I(x,y,z) reaching the point (x,y,z) from the light source. The intensity scattered along the ray to the eye is dependent on the illumination along the ray and the local density. The dependence on the density tells us that if different regions have different densities then the particles will be scattered differently. We determine the illumination by using an integral and we get the intensity of the light arriving at the eye after passing through the volume. It is implemented via voxels, it projects these voxels along a certain viewing direction and the intensities of voxels along the viewing rays are projected to provide intensity on the viewing plane.

3. Flow Diagrams

In this section we provide a basic flow chart that represents the steps involved in the algorithms.

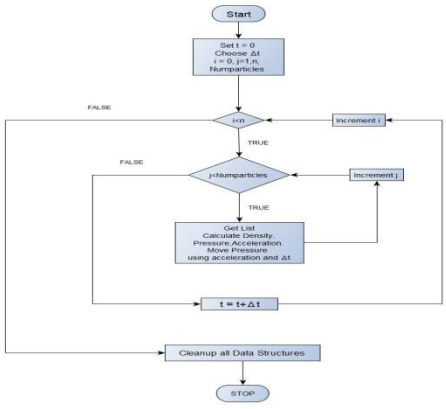

Fig. 1. Flow diagram of the particle based algorithm including the steps involved

4. Evaluation Methods

The main focus of this paper is to compare the two fluid simulations algorithms that we have defined above therefore it is very important that we make sure the evaluation procedures are correct. The procedure should be able to capture the important parameters that can define the performance of the algorithms. Also we mentioned that we would be using a complex workload so that the algorithms are tested thoroughly, in this study we used the dam break problem to be our workload. This is a standard problem in fluid simulation and have been defined time to time by many papers [8]-[10]. The problem includes simulating the flow of water as a huge tsunami wave which is then obstructed a dam wall or a solid cuboid, the resulting splashing and transfer of wave momentum is truly a challenging task and any artifacts that can occur will be visible clearly.

Now we will talk about comparison parameters, since computer graphics algorithms require a lot of components to work including the CPU, GPU and the memory buffer therefore analyzing these algorithms solely based on their time complexity is not suitable [9]. Therefore, we have used frames per second to measure the computational load of the algorithm. This is a suitable measure because the while displaying a frame onto the screen all the components of the computer are at play and if we use the same machine to run both the algorithms we can accurately measure their computational load in terms of the frames rendered by the system per second of the simulation.

For this study we will be using a Dell Vostro 3558 enterprise edition laptop with Intel core-i35005u 2 core @2.6 GHz processor and 8 gigabytes of RAM, along with an Intel HD graphics 5200 GPU with 256 MB of dedicated graphics memory buffer. The screen has a refresh rate of 60Hz and can frame rates upto 60fps, the screen resolution is 1366x768 and supports Vsync. To test the algorithm under extreme condition we will be only be using two cores to execute them so that we can test their parallelism and their utilization of multiple threads to finish the task quickly.

To test the numerical accuracy of the frames rendered we used the graphical depth test which discards out of depth samples and by looking at the results we can find whether the algorithm's numerical accuracy was satisfactory or not. The algorithms were implemented on the python and C++ programming language with use of some additional visual libraries like pygame, glu etc, the operating system they were run on was Ubuntu 18.04 LTS.As far as the 3D rendering is concerned we are using an open source software named OpenFOAM. It was created specifically for fluid dynamics simulation and has a lot of customization options, it is capable of taking in 2D grids and output a fully-fledged 3D simulation, also it provides the option to modify the transport equation and boundary conditions as required by the algorithm. OpenFOAM also provides a lot of customization options for the fluid properties like surface tension, density and turbulence. Now that we have defined the test environment we will now examine the test results.

5. Results and Discussion

We first simulated the two algorithms in a 2D environment, to generate the grids necessary for 3D simulation, we also measured the framerate and performed depth test on the rendered simulations.

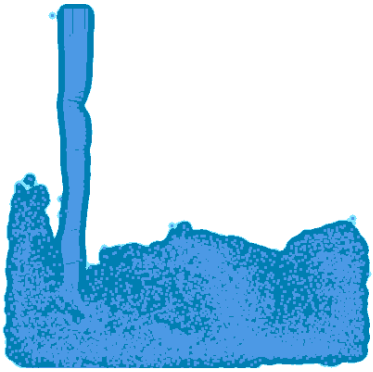

Fig. 2. Grid based 2D simulation of filling water in a glass

The grid based algorithms performs better in terms of visual aesthetics, the fluid movement is smooth and uniform but we can observe the lower framerate during the simulation. This 2D simulation is without any surface tracking the liquid particles give a fluid view by interpolating the spaces between the grid cells.

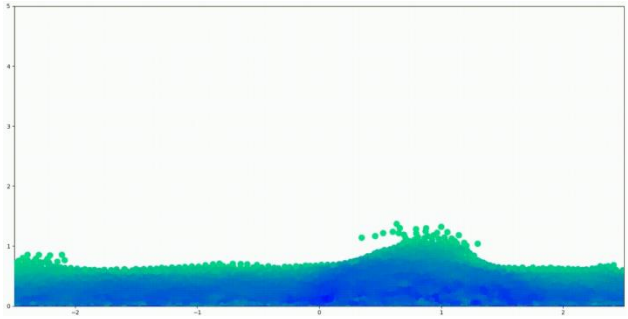

Fig. 3. Particle based 2D simulation of a standing wave

The particle based approach uses discrete blobs and we can see that we need a surface tracking algorithm to render the fluid surface because the splashing effect seems to emit small particles instead of a fluid. This the exact reason for the numerical inaccuracy of most particle based algorithms [10] but the framerate is much higher than the grid based approach.

Now that we performed the 2D simulation we will use the same for input to the OpenFOAM to render a 3D model, since we already described that we modeling the dam break problem, hence we used a cuboid as the dam wall. We monitored the framerate and used the depth test while the simulation was running to test the performance of both the algorithms.

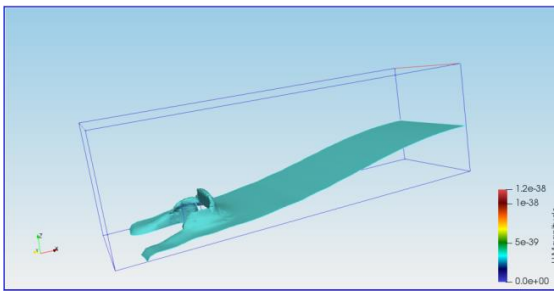

Fig. 4. OpenFOAM simulation of grid based fluid wave

With the 3D rendered model we can see that the grid based visual aesthetics are still pretty good. The grid based fluid moves more smoothly with numerical accuracy and just like the 2D simulation is flow accurate. The grid based took a heavy tool on the compute resources utilizing only about half of the CPU but rendering at a low framerate.

The particle based wave lacks just not only in numerical accuracy but also at wave motion, splashing effect is unpleasing with too much fluid scatter that looks unaesthetic, mostly due to the fact that too many particle interacting with one another will cause something known as overfitting where an algorithm computes the final result with unnecessary steps that cause the final data to look distorted but yet the framerate was very impressive for this algorithm. The algorithm exhibits excellent parallelism and was able to use 100% of both the CPU cores along with 90% GPU while rendering with 8000 particles.

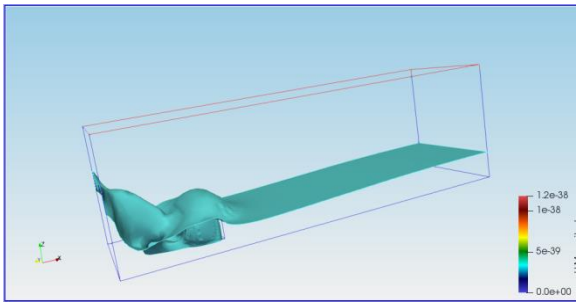

Fig. 5. OpenFOAM simulation of 3D particle based wave

Now we compare the overall results and discuss the findings. From the table give below we can see that the particle-based algorithm performed significantly better than the grid based no matter the number of particles but at lower number of particles we saw that nearly equal performance by grid and particle based approaches. the possible explanation is the use of multiple threads, when the number of particles is less then the utilization of the second core is nearly zero therefore the less parallel algorithm grid base could catch up to particle based approach but as the number of particles increase the utilization of the second core becomes an edge for particle based approaches hence they produce nearly 20% more frames than the grid based approaches.

Table 1
Framerates with respect to number of particles

No of particles	Framerate(Grid fluid)	Framerate(Particle fluid)
1000	45-50 fps	50-55 fps
4000	20-25 fps	30-40 fps
8000	5-8 fps	10-15 fps

From the result data we can see that both the algorithm performs exceptionally good when it comes to fluid simulation but the particle based approach lacks accuracy and visual aesthetics but consumes less compute power. The grid based approach is great in terms of visual aesthetics but has very less parallelism construct therefore even with a high end machine one cannot render super smooth animations while using grid

based approaches.

6. Conclusion

This study compared the performance of particle-based and grid-based fluid simulation under complex workloads. This investigation used the breaking dam problem to test the performance and graphical accuracy of the two algorithms and measured the framerate along with the particle count to realize the performance status. Based on the results obtained we saw that the grid-based algorithm was computationally expensive and consumed nearly 40% more resources than the particle-based algorithm. As far as the graphical accuracy was concerned the depth test revealed that the particle-based approach was unable to provide graphical depth to the simulation and the splashing effect was unclear with irregularly shaped waves, also the pixel overlap was high which depicted lack of numerical accuracy but the multiple core utilization was good which means it exhibits good level of parallelism.

The grid based algorithm performed well on the depth test but on the price of more computational resources, also this approach was unable to utilize the multiple cores efficiently although with a fewer number of particles the performance of particle-based and grid-based approach were nearly same. Hence we get to the conclusion that the grid-based approach is more capable of simulating the behavior of a fluid with numerical accuracy and great detail but at a high computational cost and is suitable for application like games where hardware can be upgraded as required. The particle based approach is great for small scale applications where great detail is not required like websites where hardware is limited and difficult to upgrade.

References

[1] Z. Chen, J. Qin, W. Si, T. Wong and P. Heng, "Particle-based fluid simulation with small scale details," 2014 4th IEEE International Conference on Information Science and Technology, Shenzhen, 2014, pp. 561-564.

[2] X. Shao, E. Liao and F. Zhang, "Improving SPH Fluid Simulation Using Position Based Dynamics," in IEEE Access, vol. 5, pp. 13901-13908, March. 2017.

[3] S. Li and W. Wang, "Water pouring effect simulation based on SPH," 2016 5th International Conference on Computer Science and Network Technology (ICCSNT), Changchun, 2016, pp. 94-97.

[4] Jun Chen, Kejian Yang and Yuan Yuan, "SPH-based visual simulation of fluid," 2009 4th International Conference on Computer Science & Education, Nanning, 2009, pp. 690-693.

[5] D. Mould and Y. Yang, "Modeling water for computer graphics," Computers & Graphics, vol. 21, no. 6, pp. 801–814, April. 1997.

[6] N. Chentanez and M. M¨uller, "Real-time eulerian water simulation using a restricted tall cell grid," ACM Transactions on Graphics, vol. 30, no. 4, pp. 82:1–82:10, September. 2011.

[7] P. Kipfer and R. Westermann, "Realistic and interactive simulation of rivers," Proceedings of Graphics Interface 2006, pp. 41–48.

[8] A. T. Howes and A. R. Forrest, "Visual simulation of waterfalls and other water phenomena," Proceedings of the ACM SIGGRAPH 97 Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '97, p. 146, 1997.

[9] M. B. Nielsen and O. Østerby, "A two-continua approach to eulerian simulation of water spray," ACM Transactions on Graphics, vol. 32, no. 4, pp. 67:1–67:10, December. 2013.

[10] J. Stam, "Stable fluids," Proceedings of the ACM SIGGRAPH 99, pp. 121–128, March. 1999.