Adaptive Cloud Simulation using Position Based Fluids

Abstract

In this paper, we propose a method for the simulation of clouds using particles exclusively. The method is based on Position Based Fluids, which simulates fluids using position constraints. To reduce the simulation time, we've used adaptive splitting and merging to concentrate computation on regions where it is most needed. When clouds are formed, particles are split so as to add more detail to the generated cloud surface and when they disappear, particles are merged back. We implement our adaptive method on the GPU to accelerate the computation. While the splitting portion is easily parallelizable, the merge portion is not. We develop a simple algorithm to address this problem and achieve reasonable simulation times.

Keywords: cloud simulation, particle-based, adaptive simulation

1 Introduction

Outdoor scenes are commonly used in movies and games, and an important element of such scenes is clouds. While a clear sky is beautiful on its own, clouds give depth to the sky.

Cloud generation methods can be divided into two large categories: procedural and simulation methods. Procedural methods such as [1, 2], try to generate clouds in real-time with the shape controlled by the user. However, finding the best parameters is a trial and error process, which can be an issue. On the other hand, simulation methods such as [3, 4] use physically based equations to move the underlying fluid and generate clouds.

In this paper, we propose a simulation method using particles based on Position Based Fluids [5] for the generation of clouds. For the cloud generation, we use a heating and cooling process based on the one used in [4]. However, only this is not sufficient for generating clouds.

A difficulty seen with simulating clouds with particles is that the particles do not share their temperature and cloud density with each other. As a result, particles in isolation will turn into clouds but it will not have a meaningful shape. To solve this, we propose a smoothing method similar to Position Based Dynamics [6].

We've also developed an adaptive method based on [7]. We propose to use each particle's cloud density to detect areas of interest. Particles are split in areas of most interest and merged together in areas of least interest.

The system was implemented on the GPU using CUDA®. One particular issue we've found in this implementation is that the merge algorithm is not inherently parallel, and we show how we've implemented it in the GPU.

The rest of the paper is organized as follows. In section 2 some related work in the area of fluid and cloud simulation is shown. In section 3 we show some basics of Position Based Fluids. In section 4 we present the cloud generation process, how it is discretized using particles and how the smoothing method works. In section 5, some details such as periodic boundary conditions and parallel adaptive algorithm are shown. In section 6, results and comparisons of the method are shown. Finally, we conclude the paper in section 7.

2 Related work

Cloud generation methods in computer graphics can be divided into procedural methods and simulation methods. Procedural methods seek

to generate a cloud-like shape, where this shape is controlled by the user. When possible, these methods try to generate the clouds in real-time. These methods include works based on noise functions [1], fractals [8], interactive design [2, 9, 10], and creation from single images [11, 12].

Simulation methods, on the other hand, seek to generate physically accurate clouds. Miyazaki et al. [3] used a grid-based simulation [13] to simulate the underlying fluid, and used differential equations to describe the cloud generation process. Harris et al. [14] proposed a similar method, but used slightly different definitions for the same process. While not a new simulation method, Dobashi et al. [4] proposed a control method based on [3], for the generation of clouds that have a particular shape.

Particles methods for fluid simulations are usually based on Smoothed Particle Hydrodynamics(SPH) [15]. In this, the Navier-Stokes equation is discretized and solved using the particles exclusively. While in grid-based methods such as [13], a linear system of equations is solved to keep the fluid incompressible, an equation of state is used to calculate the fluid pressure in [15]. To improve on this, a number of methods have been proposed to simulate incompressible fluids using SPH [16, 17].

Macklin et al. [5] proposed Position Based Fluids based on Position Based Dynamics [6] that doesn't require to solve the Navier-Stokes equation, and as a result, larger time steps can be used and less computational time is required to simulate incompressible fluids. We've used this method as the basis for our simulation, but it is not dependent on it, i.e, we can use any particle method for the simulation. In the next section, we describe some more details of Position Based Fluids.

3 Position Based Fluids

Position Based Fluids(PBF) [5] is a particle-based method that is capable of simulating incompressible fluids, such as water and smoke [18]. Instead of solving the Navier-Stokes equation such as in [15] or solving linear system of equations such as in [17], PBF uses position constraints to guarantee the fluid's density doesn't change. The constraints are then solved similar to Position Based Dynamics(PBD) [6].

Similarly to SPH [15], the density of each particle is calculated as a weighted sum of the neighboring particles:

$$\rho_i = \sum_j m_j W_{i,j}, \tag{1}$$

where ρ_i is the density of particle i, j are the neighboring particles, m is the mass of each particle and $W_{i,j}$ is a normalized weight function, such as the one from [15]. A position constraint is defined for each particle so as the density doesn't change:

$$C_i(\mathbf{x}) = \frac{\rho_i}{\rho_0} - 1, \tag{2}$$

where C is the position constraint, $\mathbf{x}$ is position vector for all particles and ρ_0 is the rest density of the fluid. When this constraint is zero for each particle, we're able to guarantee the fluid's density doesn't change. However, because this is a constraint based on each particle's density and position, this is a non-linear constraint. To solve such a constraint, PBD linearizes it by taking the Taylor's expansion and using only the linear component [6]. The expanded constraint is:

$$C_i(\mathbf{x} + \Delta\mathbf{x}) = C_i(\mathbf{x}) + \Delta\mathbf{x} \cdot \nabla C_i(\mathbf{x}), \tag{3}$$

where $\Delta\mathbf{x}$ is the change vector for all particles, i.e, how they need to move after the constraint is solved. By supposing the change vector is in the same direction as the constraint's gradient, we can finally solve the constraints:

$$C_i(\mathbf{x}) + \lambda_i \nabla C_i(\mathbf{x}) \cdot \nabla C_i(\mathbf{x}) = 0, \tag{4}$$

where we are interested in solving λ for each particle i. The particles can finally be moved as described in [5].

4 Cloud dynamics

Our simulation method is based on the one proposed in [3, 4]. In their method, a grid based simulation is used for the simulation of clouds. We propose to apply their method to particles, and show what needs to be done to achieve good results.

Each particle tracks its temperature, vapor density and cloud density. At first, the particles are moved as described in section 3. Next,

the temperature field is smoothed as described in 4.1. Next, the particles near the ground are heated up:

$$\frac{dT_i}{dt} = S_e, \tag{5}$$

where T_i is the temperature of particle i, and S_e is the external heat source. Next, buoyancy and gravity forces are applied to each particle:

$$\mathbf{F}_i = k_b \frac{T_i - T_0}{T_0}\mathbf{j} - k_g w_i \mathbf{j}, \tag{6}$$

where $\mathbf{F}_i$ is the result force of particle i, k_b and k_g are user-defined buoyancy and gravity constants, respectively, T_0 is the environment temperature at the particle position, w_i is the cloud's density of particle i, and $\mathbf{j}$ is the upward direction vector. In our system, the environment temperature is defined as a linear transition from the bottom to the top of the domain, where the highest temperature is in the bottom of the domain.

When the particle rises, it cools down due to adiabatic cooling. This process is described by:

$$\frac{dT_i}{dt} = -\Gamma_d u_y, \tag{7}$$

where Γ_d is the dry adiabatic lapse rate and u_y is the particle's vertical velocity. Next, the vapor and cloud densities of each particle are updated according to the temperature. This process, called phase transition, is described by:

$$\frac{dw_i}{dt} = -\frac{dv_i}{dt}, \tag{8}$$

where v_i is the vapor's density for particle i. We discretize this equation similarly to [3]:

$$\frac{dw_i}{dt} = \alpha(v_i - \gamma_i) \tag{9a}$$

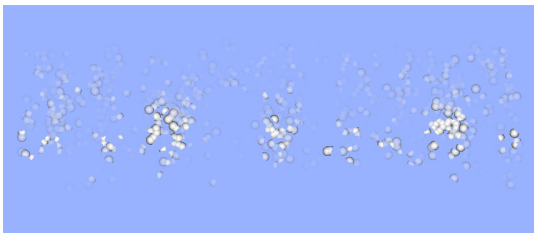
$$\frac{dv_i}{dt} = -\alpha(v_i - \gamma_i), \tag{9b}$$

where α is the phase transition rate and γ_i is the vapor's density upper limit for particle i. This upper limit is calculated as:

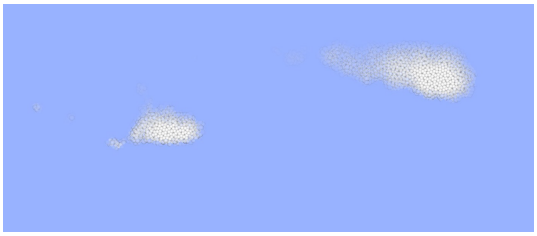
$$\gamma_i = \min(A \exp(\frac{B}{T_i + C}), v_i + w_i), \tag{10}$$

where A, B and C are user-defined phase transition parameters. When the vapor turns into cloud, it releases latent heat, and the particle's temperature increases:

$$\frac{dT_i}{dt} = Q\alpha(v_i - \gamma_i), \tag{11}$$



(a) Non-smoothed result



(b) Smoothed result

Figure 1: Two-dimensional results showing difference between smoothed and non-smoothed scalar fields. Particles are shown as spheres and transparency is related to the cloud's density.

where Q is the latent heat coefficient. While this should be sufficient to simulate clouds, this doesn't generate good results. Figure 1a demonstrates one such example in two-dimensions.

4.1 Smoothing of scalar fields

The reason for the issue shown in Figure 1a is because the scalar fields (temperature, cloud density and vapor density) are not smooth in the whole domain, i.e, the difference between two nearby particles is too large. We propose a method to smooth the temperature field using constraints similar to PBF. Figure 1b shows a two-dimensional result when using our method.

First, we define a field constraint for each particle:

$$C_i(\mathbf{T}) = \nabla^2 T_i, \tag{12}$$

where $\mathbf{T}$ is the vector of temperature values of all particles.

When the above constraint is equal to 0, it means the temperature near particle i is smooth. By applying the constraint to all particles, we're able to smooth the temperature in the whole domain. We've discretized the above laplacian us-

ing SPH [19]:

$$\nabla^2 T_i = \sum_j \frac{T_i - T_j}{\rho_0} \frac{\mathbf{x}_{i,j}}{|\mathbf{x}_{i,j}|^2 + \epsilon} \cdot \nabla W_{i,j}, \quad (13)$$

where $\mathbf{x}_{i,j} = \mathbf{x}_i - \mathbf{x}_j$, $\mathbf{x}_i$ is the position vector for particle i, and ϵ is a small constant used to avoid division by 0.

Similarly to PBF, we linearize the above constraint by taking the Taylor's expansion:

$$C_i(\mathbf{T} + \Delta\mathbf{T}) = C_i(\mathbf{T}) + \Delta\mathbf{T} \cdot \nabla_T C_i(\mathbf{T}), \quad (14)$$

where $\Delta\mathbf{T}$ is the change in the temperature field. By supposing the change is proportional to the constraint gradient, we have:

$$\Delta\mathbf{T} = \lambda_i \nabla_T C_i(\mathbf{T}) \quad (15a)$$

$$C_i(\mathbf{T}) + \lambda_i |\nabla_T C_i(\mathbf{T})|^2 = 0. \quad (15b)$$

The constraint gradient is a vector with N components(the number of particles) where for each index k we have:

$$\nabla_T C_{i,k}(\mathbf{T}) = \begin{cases} \sum_j \frac{\mathbf{x}_{i,j}}{\rho_0 |\mathbf{x}_{i,j}|^2 + \epsilon} \cdot \nabla W_{i,j} & \text{if } k = i \\ -\frac{\mathbf{x}_{i,k}}{\rho_0 |\mathbf{x}_{i,k}|^2 + \epsilon} \cdot \nabla W_{i,k} & \text{otherwise} \end{cases} \quad (16)$$

which is simply the derivative on the temperature field for each index. We can finally solve for λ_i in equation (15b) and update the particle's temperature:

$$\lambda_i = -\frac{\nabla^2 T_i}{|\nabla_T C_i(\mathbf{T})|^2} \quad (17a)$$

$$T_i^* = T_i + \kappa \lambda_i \nabla_T C_{i,i}(\mathbf{T}), \quad (17b)$$

where T_i^* is the smoothed temperature value, κ is a small constant (taken to be equal to 0.1 in our experiments) and $\nabla_T C_{i,i}(\mathbf{T})$ is calculated as in equation (16). While we want to smooth the temperature field, if we use κ equal to 1, the temperature field will be too smooth, and the result clouds will lose some detail, hence a smaller κ value is used.

5 Implementation

In this section, we describe some implementation details of our system, such as how we've implemented periodic boundary conditions and the adaptive algorithm for splitting and merging of particles. Our system was implemented in CUDA® and we also show some considerations for an implementation on the GPU.

We define a fixed domain and create fluid particles uniformly in this domain. Their velocity is initialized to a random direction and maximum speed to a user-defined constant. Their temperature is initialized to the environment temperature T_0. Their cloud density is initialized to 0 and vapor density is initialized to $k_v A \exp(\frac{B}{T_i + C})$, where k_v is a user-defined constant(taken to be 0.5 in our tests), and A, B and C are the phase transition parameters defined in equation (10).

5.1 Periodic boundary conditions

Clouds simulations commonly use periodic boundary conditions to achieve a constant flow [3, 4]. A property of periodic boundary conditions is that particles in one side of the domain are influenced by particles in the other side of the domain.

To achieve such an effect, we've used image particles in the periodic portions of the domain and boundary particles in the non-periodic portions. In our system, we've taken the bottom and top of the domain to be walls, and the x and z directions to be periodic.

In the beginning of the simulation, we create boundary particles which will be used throughout the simulation. These are created near the walls of the domain, with a constant spacing of d, the particle spacing and h deep, the SPH kernel radius. This way, a fluid particle near the walls should have a full neighborhood. Figure 2 illustrates the domain and boundary particles.

Enough space for all image particles is allocated and reused throughout the simulation. For instance, a particle in the corner of the domain needs to be mirrored in the x direction, in the z direction and in both directions at once: we need at most three times the number of particles to store all image particles. A fluid particle is mirrored only if it is within a distance of h from the limits of the domain, i.e, if it would influence a particle in the other side. To save in memory usage, each image particle only stores its position and a pointer to the original fluid particle.

As usual in SPH and PBF implementations, we maintain three neighborhood lists for each

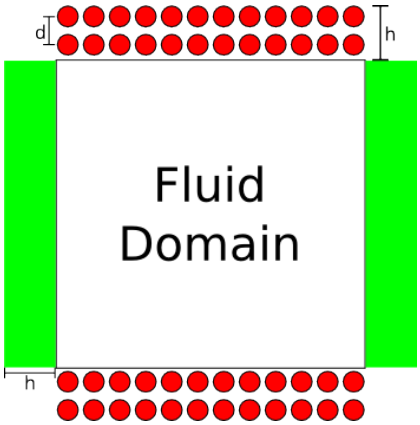


Figure 2: Two-dimensional representation of the fluid domain, periodic regions(green) and boundary particles(red).

particle: fluid neighbors, image neighbors and boundary neighbors. Every neighbor is used in simulation calculations (both PBF and scalar field smoothing).

In the case of image neighbors, the values of the original particle (except for the position) are used in density and constraint calculations. In the case of boundary neighbors, it requires a mass and temperature.

The temperature of the boundary particles is initialized the same way a fluid particle in the bottom (or top) of the domain would. The mass is required only in the density of a fluid particle, which can be written in its final form as:

$$\rho_i = \sum_j m_j W_{i,j} + \sum_k m_i W_{i,k}, \qquad (18)$$

where ρ_i is the density of the fluid particle i, j are the fluid and image neighbors, and k are the boundary neighbors. We could have defined the boundary particles' mass in a similar manner to [20], but we didn't find it necessary, as the initial spacing is the same for both fluid and boundary particles.

5.2 Adaptive particles

One property of particle-based simulations is that adaptively increasing the number of particles in areas of interest can be achieved by simply splitting and merging nearby particles. We've based our adaptive algorithm on [7], which detects areas of interest, and splits particles into two smaller ones in areas of interest, or merge two particles into larger ones in areas of little interest.

Similarly to [7], each particle stores its adaptive level. The adaptive level is initialized to 0, increased when the particle is split and decreased when the particle is merged. To preserve the mass conservation property of the system, the radius of the particle is defined as $h_i = h_0 \cdot 2^{-\frac{l}{3}}$, and the mass of the particle is defined as $m_i = m_0 \cdot 2^{-l}$, where l is the adaptive level, h_0 is the initial SPH kernel radius and m_0 is the initial mass.

When particles have different radius, the SPH kernels need to be changed as well to preserve the accuracy of the kernel. We found the following to give good results when using PBF:

$$\hat{W}_{i,j} = \frac{1}{2}\left(W_{i,j}(h_i) + W_{i,j}(h_j)\right) \qquad (19a)$$

$$\nabla\hat{W}_{i,j} = \frac{1}{2}\left(\nabla W_{i,j}(h_i) + \nabla W_{i,j}(h_j)\right) \qquad (19b)$$

where $W_{i,j}(h_i)$ is the original SPH kernel applied to the radius h_i, and $\hat{W}_{i,j}$ is the new SPH kernel that is used in all calculations such as PBF constraints and smoothing of scalar fields.

We define areas of interest based on the particle's cloud density. For each particle, a shape energy E is defined and initialized to 0. The shape energy is what defines if a particles needs to be split or merged. At the end of every step, the shape energy is updated using the following:

$$E = k_p(1 - \exp(-k_c w_i)), \qquad (20)$$

where k_c and k_p are user-defined constants, and w_i the particle's cloud density. Clouds have multiple parts to them: dense and not so dense parts. Using the density directly makes it difficult to control splitting of particles on the surface of the cloud. However, using the above shape energy function, particles on the surface of the cloud have a higher energy due to the exponential, and this, in turn, becomes easier for the user to control if they are to be split or not.

Next, particles with a shape energy value greater than α, are marked to be split, and particles with a shape energy value lower than β are marked to be merged, where α and β are user-defined constants. Then, the split and merge algorithms of the following sections are run.

We've run an experiment to verify if the above shape energy is good enough to identify areas of

interest. Figure 3 demonstrates this result. In this experiment, we've run a two-dimensional simulation and visualized the particle's shape energy. Particles that have become clouds have a high energy. Particles that have ceased to be clouds, have a low energy and are candidates for merging.

As a comparison, we've visualized the cloud density similarly to the shape energy, i.e, what would happen if we used the cloud density directly as the shape energy. While it can detect the regions of interest, some detail is clearly being lost: some regions of the cloud are not dense, but are still good candidates for splitting.

5.2.1 Splitting

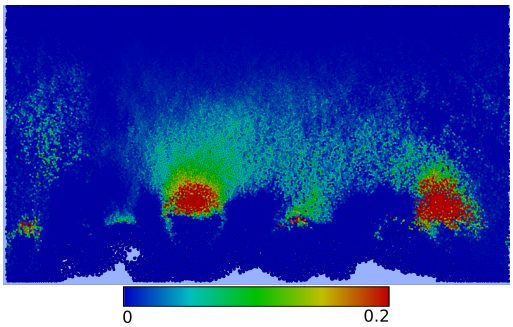
Particles marked to be split are checked if they can be split into two new particles. The properties of the new particles are the same as the original particle and they are spaced equally from the original one. To avoid excessive forces due to the new spacing, the particle is only split if the new particles are not too close to other existing ones. For more details on the splitting algorithm, see [7].

To properly parellelize the algorithm, we allocate enough memory in the beginning of the simulation to accomodate a number of split particles. The splitting algorithm is divided into two parts. First, the above constraints are verified and the number of split particles is calculated. Since the split particle can be removed, it is reused for one of the new split particles. To define the position in memory for the second particle, we use prefix sum [21]. The second part updates the particles in memory.

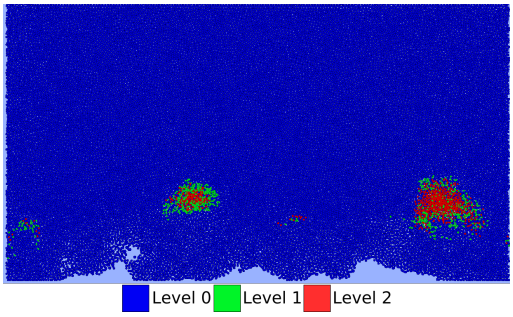
5.2.2 Merging

If two particles are neighbors, share the same level and are marked to be merged, they are tentatively merged into a new one. The properties of the new particle is the average of the original ones and it is placed in the middle between the two old particles. Also, it is only created if there are no other particles too close to the new one [7].

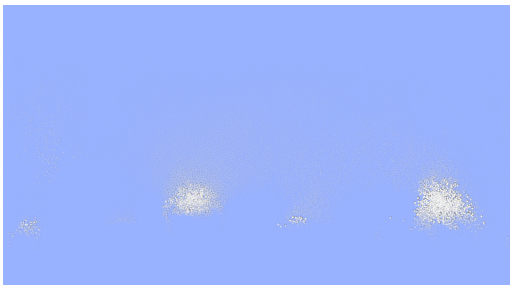
To properly implement this on the GPU, we need to guarantee that two particles to be merged are merged between themselves. However, in the naïve implementation, synchronization issues may happen. For instance, suppose that three distinct particles i, j and k are to be merged, and that they are neighbors. Without care, it is possible that particle i merges itself

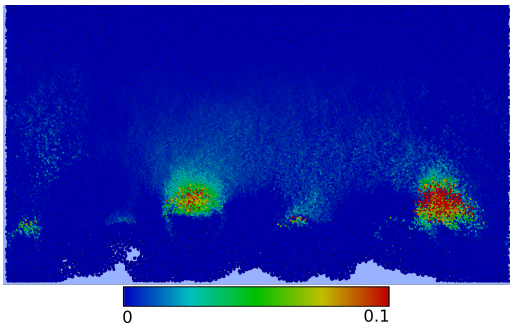


(a) Shape energy



(b) Adaptive level


(c) Cloud result



(d) Cloud density as energy

Figure 3: Result of the shape energy experiment. Approximately 40,000 particles were used.

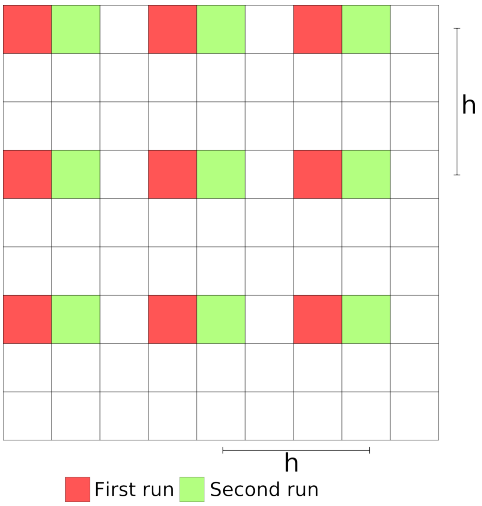


Figure 4: Illustration of the thread allocation on the grid for the parallel merge algorithm. Grid cells of the same color are checked in the same run. Cells of different colors are neighbors of each other and must be checked in different runs. The white cells are checked in subsequent runs.

with particle j, but particle j merges itself with particle k, resulting in wrong particle data. This obviously can be solved if the threads use global communication and mutual exclusion. But instead of using global communication, we've implemented this based on the underlying grid information.

To accelerate neighborhood search, we've used a uniform grid similar to the index sort grid shown in [22]. In this method, a linear index is attributed to each particle based on their position on this grid. For each grid cell, we verify the particles inside this grid cell and merge particles as necessary.

To guarantee that the above three-particle synchronization problem doesn't occur, we need to check particles in such a way that two neighbor particles are not checked at the same time. To do that, instead of creating threads on the particles, we create threads on the underlying uniform grid.

For any two threads, the grid cells of each thread must not be neighbor of each other. Two grid cells are not neighbors if any particle in the first cell is not a neighbor of any particle in the second cell. In other words, the checked cells must be a minimum distance of d of each other (in number of cells):

$$d = \left\lceil \frac{h}{L} \right\rceil , \tag{21}$$

where h is the maximum possible SPH kernel radius and L is the grid cell side length. As a result, multiple runs of the algorithm must be done to check the whole underlying grid. In figure 4, the thread allocation on the grid is illustrated.

6 Results

In the following, we will describe the examples we've run and compare the simulation times. Our simulation hardware was a machine with a Intel® Core™i7-4770k CPU, 16GB of memory and a NVIDIA® GeForce® GTX 780 Ti GPU. The final renderings were done using mental ray [23].

In the first example, we did not apply gravity to the whole domain except for the one described in equation (6). The ground is heated in a random pattern using procedural noise [24] during the whole simulation. For this reason, the clouds grow without any limitation.

In the second example, not only the gravity's force due to the cloud density is applied, but a constant gravity in the whole domain is also applied. As a result, the clouds constantly disappear and reappear. Figure 5 shows frame stills for both examples.

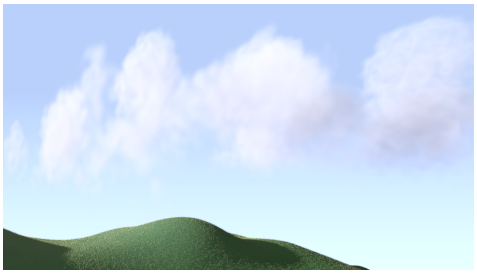
The simulation times for the adaptive and non-adaptive versions of the examples were compared. Table 1 shows the results. We've set the examples in such a way that a particle of a higher level in the adaptive version would have a similar size to the non-adaptive version, i.e, the clouds have particles of similar size but not in the rest of the domain.

The non-adaptive versions have more particles and, as a result, more time is necessary to simulate them. In the adaptive versions, even though we're using less particles, the visual results are similar. We can also see from the results that the proposed splitting and merging algorithms are fast enough that they do not interfere too much in the simulation.

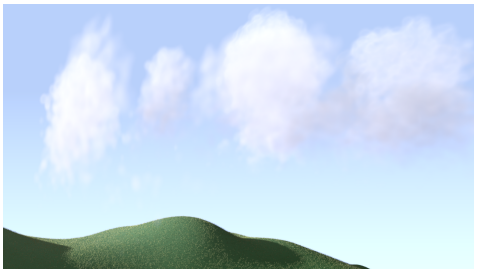
The visual results do differ though, and we believe this to be a limitation of PBF. As noted by Macklin et al. [5, 18], their method is sensible to changes in the simulation resolution and

Example	Particle number	PBF	Smoothing	Cloud dynamics	Splitting	Merging
Fig. 5a	840k	1087.36	112.35	19.24	-	-
Fig. 5b	540k	628.78	64.55	12.50	6.10	23.71
Fig. 5c	930k	1291.85	137.00	20.06	-	-
Fig. 5d	500k	748.81	71.94	13.78	6.41	24.63

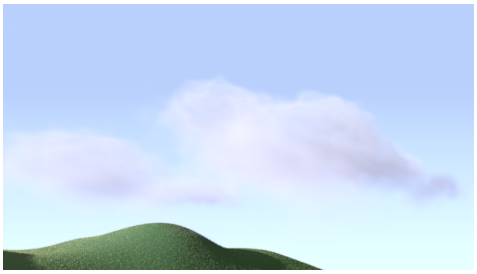
Table 1: Average time spent in each phase of the method for each example. Times in milliseconds. Particle number is the maximum number of particles in the whole simulation.



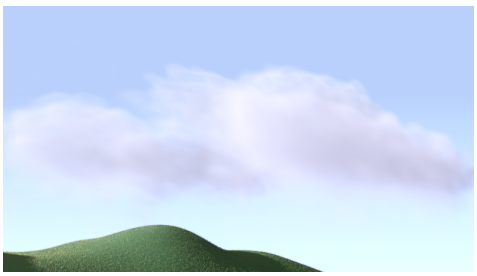
(a) Frame 700 of the first example (Non-adaptive)



(b) Frame 700 of the first example (Adaptive)



(c) Frame 500 of the second example (Non-adaptive)



(d) Frame 500 of the second example (Adaptive)

Figure 5: Results of the proposed method.

we verify that here. When particles are split or merged, they introduce small forces in the neighboring particles, mostly due to their sudden movement. However, this doesn't happen in the method of Adams et al. [7] when using SPH.

7 Conclusion

In this paper, we've proposed a particle based cloud simulation method based on Position Based Fluids [5]. To guarantee that the generated clouds make sense, we smooth the temperature field with a method similar to Position Based Dynamics [6].

To save on computational time, we've implemented the proposed method on the GPU using CUDA®. We've also implemented a adaptive method on the GPU based on [7]. The generated clouds are similar to previous grid based simulation methods, which is what we expected.

In future work, investigating how PBF can be applied to particles of varying sizes is very important for generating high quality images. Also, a control method similar to [4] is desirable.

References

[1] David S. Ebert. Volumetric modeling with implicit functions: A cloud is born. In *ACM SIGGRAPH 97 Visual Proceedings: The Art and Interdisciplinary Programs of SIGGRAPH '97*, SIGGRAPH '97, pages 147–, New York, NY, USA, 1997. ACM.

[2] Jamie Wither, Antoine Bouthors, and Marie-Paule Cani. Rapid sketch modeling of clouds. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'08, pages 113–118, Aire-la-Ville, Switzerland,

Switzerland, 2008. Eurographics Association.

[3] Ryo Miyazaki, Yoshinori Dobashi, and Tomoyuki Nishita. Simulation of cumuliform clouds based on computational fluid dynamics. *Proc. Eurographics 2002 Short Presentation*, pages 405–410, 2002.

[4] Yoshinori Dobashi, Katsutoshi Kusumoto, Tomoyuki Nishita, and Tsuyoshi Yamamoto. Feedback control of cumuliform cloud formation based on computational fluid dynamics. *ACM Trans. Graph.*, 27(3):94:1–94:8, August 2008.

[5] Miles Macklin and Matthias Müller. Position based fluids. *ACM Trans. Graph.*, 32(4):104:1–104:12, July 2013.

[6] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, April 2007.

[7] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[8] Richard Voss. fourier synthesis of gaussian fractals: 1f noises, landscapes, and flakes. *State of the Art in Image Synthesis Tutorial Notes*, 10, 1983.

[9] Joshua Schpok, Joseph Simons, David S. Ebert, and Charles Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 160–166, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[10] Antoine Bouthors, Fabrice Neyret, et al. Modeling clouds shape. In *Eurographics (short papers)*, 2004.

[11] Yoshinori Dobashi, Wataru Iwasaki, Ayumi Ono, Tsuyoshi Yamamoto, Yonghao Yue, and Tomoyuki Nishita. An inverse problem approach for automatically adjusting the parameters for rendering clouds using photographs. *ACM Trans. Graph.*, 31(6):145, 2012.

[12] Chunqiang Yuan, Xiaohui Liang, Shiyu Hao, Yue Qi, and Qinping Zhao. Modelling cumulus cloud shape from a single image. *Computer Graphics Forum*, 33(6):288–297, 2014.

[13] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[14] Mark J. Harris, William V. Baxter, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '03, pages 92–101, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[15] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[16] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible sph. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 40:1–40:6, New York, NY, USA, 2009. ACM.

[17] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible sph. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):426–435, March 2014.

[18] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim.

Unified particle physics for real-time applications. *ACM Trans. Graph.*, 33(4):153:1–153:12, July 2014.

[19] Fabiano Petronetto, Afonso Paiva, Marcos Lage, Geovan Tavares, Helio Lopes, and Thomas Lewiner. Meshless helmholtz-hodge decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):338–349, 2010.

[20] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible sph. *ACM Trans. Graph.*, 31(4):62:1–62:8, July 2012.

[21] Mark Harris, Shubhabrata Sengupta, and John D. Owens. Parallel prefix sum (scan) with cuda. In Hubert Nguyen, editor, *GPU Gems 3*. Addison Wesley, August 2007.

[22] Markus Ihmsen, Nadir Akinci, Markus Becker, and Matthias Teschner. A parallel sph implementation on multi-core cpus. *Computer Graphics Forum*, 30(1):99–112, 2011.

[23] NVIDIA ARC. mental ray [software], 1 2015.

[24] Stefan Gustavson. Simplex noise demystified. *Linköping University, Linköping, Sweden, Research Report*, 2005.