STAG: Smart Tools and Applications in Graphics (2019)
M. Agus, M. Corsini and R. Pintus (Editors)

Interactive animation of single-layer cumulus clouds using cloud map

Prashant Goswami[†]

Blekinge Institute of Technology, Karlskrona (Sweden)

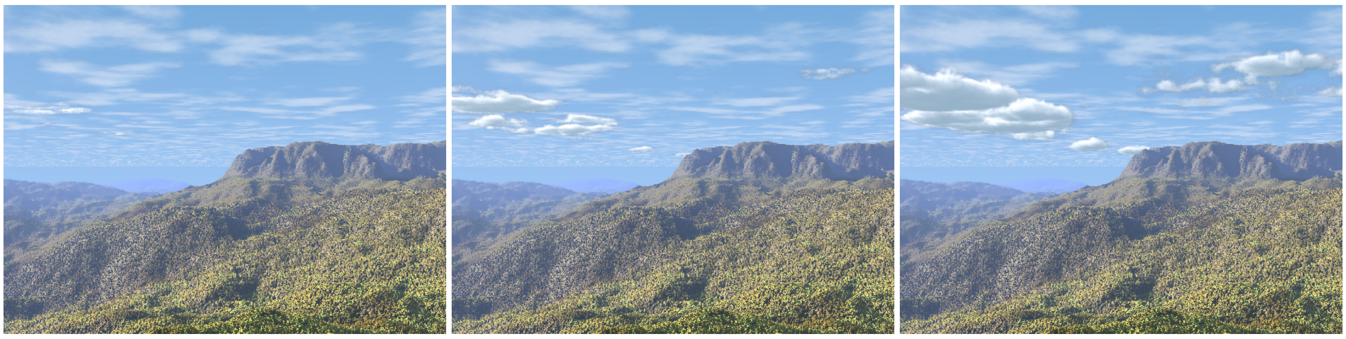


Figure 1: *Landscape-scale cumulus cloud animation using our approach for the single-layered cumulus clouds, simulation together with the rendering running at around 200 fps.*

Abstract

In this paper, we present a physics-driven procedural method for the interactive animation of realistic, single-layered cumulus clouds for the landscape-scale size. Our method employs the coarse units called parcels for the physics simulation and achieves procedural micro-level volumetric amplification based on the macro physics parameters. However, contrary to the previous methods which achieve amplification directly inside the parcels, we make use of the two-dimensional texture called cloud maps to this end. This not only improves the shape and distribution of the cloud cover over the landscape but also boosts the animation efficiency significantly, allowing the overall approach to run at high frame rates, which is verified by the experiments presented in the paper.

CCS Concepts

• ***Computing methodologies*** → ***Animation;*** *Procedural animation;*

1. Introduction

Clouds form an important component of several background scenes in the video games and other interactive applications, many of which demand dynamic clouds with evolving shapes. Whereas impressive static images of the clouds can be obtained using purely procedural techniques [MMPZ12], realistic cloud animation at real-time rates remains a challenge. Methods in the computational fluid dynamics (CFD) are dependent on the accurate initial conditions to simulate cloud development [Mih11]. Moreover, they are computationally intensive and not suitable for interactive applications. In computer graphics, most of the methods that deal with the physics-based simulation of clouds are offline. Other methods often need high resolution to produce a limited volume of clouds online [FBDY15]. Furthermore, all of them require a separate pass for the surface generation and rendering which is often offline, leaving them unsuitable for games.

Recently hybrid methods [GN17] are proposed, where a physics-inspired procedural technique is used for the real-time cloud animation. This is achieved by using coarse primitives (parcels) at the macro-level to compute the physics followed by the hypertexture generation at the micro-level inside these parcels for cloud visualization. This significantly reduces the computational time spent on simulating the underlying physics which otherwise is a very ex-

[†] prashant.goswami@bth.se

DOI: 10.2312/stag.20191365

pensive component. As a second step, these macro components can be raytraced at the real-time frame rates using procedural amplification, which itself is tuned as a function of the macro variables computed from the physics.

In this paper, we improve on their method for animating single-layered cumulus clouds, in terms of appearance and efficiency. The motivation for this improvement comes from the fact that in nature, a large type of cumulus clouds of varying shapes and sizes are present in a single layer with minor vertical displacements between them [Day05]. To this end, we employ *cloud map*, a bounded 2D texture which guides cloud formation and provides us a handle to generate more natural cloud shapes efficiently.

The key contributions of our work are:

- To employ a 2D cloud map for the single-layer cumulus cloud animation to support a more realistic cloud cover and evolving cumulus cloud shapes
- To simplify and reduce the computational overhead of physics and rendering, by operating them on the cloud map domain instead of a much larger free space

Our approach leveraging the cloud maps not only inherits a better cloud shape and distribution over the landscape, but also produces an efficient interactive animation that could be easily used in the games or flight simulators as a background component on the landscape-scale terrain size. In contrast to the existing approach where volume amplification is achieved directly inside the parcels, our method disentangles the physics and the rendering part. The cloud cover is obtained from the cloud map which eliminates need for the user to specify the parcel distribution.

2. Related Work

Cloud generation is an extensively studied research topic in the field of computer graphics for the past several decades. Researchers have looked into the various aspects of cloud generation including physics, shape modeling and rendering. [KVH84] trace densities within a volume grid and to this end, they review dynamical models of clouds. [BN04] use blobs to model the shape of static cumulus clouds. Recently, [KMM*17] use neural networks to synthesize clouds images with enhanced illumination.

Some early work has been targeted to achieve simplified animation. In [HBSL03] voxel-based Eulerian physics is used to simulate the cloud evolution. In [DKY*00] the dynamics of cloud formation is simplified and simulated using a cellular automata. The obtained density is then rendered with the splatting algorithm. Bi et al. [BBZ*16] have combined cellular automata together with the particle system to achieve the cloud modeling and visualization. Earth-scale clouds for photorealistic synthesis of images taken from the space are animated by specifying centers of high and low pressure in [DYN06]. [GD07] employ particle-based cloud simulation using the basic stability criteria for the air. [FBDY15] propose adaptive particle-based cloud simulation using the position based fluids on the GPU [MM13], where the particles are split and merged to concentrate computation on regions where it is most needed.

In [YW11], cloud modeling, rendering and morphing is attained with the help of the 2D images. Recently a purely procedural animation for cloudspaces is presented in [WCGG18]. Their method is capable of simulating various types of clouds. Instead of using physics-based simulation, they animate cloudscapes by morphing between the pre-selected key-frames with the help of user control. [Ney97] use higher level guiding variables to obtain convective cloud formation whereas [SSEH03] propose an interactive system to artistically model and animate the cloud systems. [SMST] model the cumulus cloud formation with the help of mass-flux of turbulent updraughts. In [DG17] a real-time, CPU-based cloud simulator is proposed that uses available sounding data to simulate 3D clouds. Their results are impressive for the limited landscape sizes where a high frame rate is not required.

It is worth noting that a majority of the aforementioned work focuses on a single aspect of cloud generation: modeling, animating or visualization. Some of the techniques can achieve offline visualization in addition to modeling or animation. However, none of them can be used for the real-time animation together with the visualization. Furthermore, they are not extensively scalable since they model a limited volume of the atmosphere for the simulation purpose.

[GN15, GN17] introduced the first hybrid model that combines procedural rendering with physics for the real-time animation of cumulus clouds. Though their method is capable of achieving interactive frame rates for a few parcels, the performance drops quickly as the number of parcels grow. In their method, one either requires few big or several small parcels to produce a good cloud coverage on the landscape. In both of these cases, the rendering performance drops significantly as one intends to increase the cloud coverage. Furthermore, the generated clouds are limited in their shapes and distribution over the landscape which has to be controlled explicitly. We present an improvement of their model for simulating and rendering single-layered cumulus clouds. Our method is capable of generating a more realistic, natural looking cloud cover and animate it efficiently for real-time applications. We expand more on this extension and extensively describe our approach in Sec.3, followed by results in Sec. 4.

3. Method

We begin with a brief overview of the underlying physics by Goswami and Neyret [GN17] which forms the basis of our approach.

3.1. Base physics

The atmosphere is implicitly represented with a temperature and humidity profile varying with the altitude. Physics is carried out at the macro level within a bounded domain using octree in the spherical thermal units called parcels, while the rendering is achieved with the hypertexture on-the-fly at the micro level inside the simulated parcels (see Fig. 2). The buoyant force is obtained with the Archimedes principle using the difference of virtual potential temperature of parcel (Θ^{parcel}) with that of the surrounding environment (Θ^{atm}):

$$\mathbf{f}_{buoy} \approx (\frac{\Theta_v^{parcel} - \Theta_v^{atm}}{\Theta_v^{atm}})g \quad (1)$$

The interaction between the parcels is handled using the smoothed particle hydrodynamics (SPH) where pressure and the viscosity forces come into play. These parcels interact with the implicit environment through the process of entrainment and detrainment which causes mixing of the parcel content with the surrounding atmosphere. During entrainment, the surrounding relatively drier air enters and mixes with the parcels whereas detrainment refers to the process of parcels ejecting their moisture content slowly to the atmosphere.

The parcels are randomly created in the regions containing a higher humidity in 3D space, which rise due to the positive buoyant force (initialized parcels are slightly warmer than the surrounding air), where the difference in virtual potential temperature is used to compute net buoyant force on a parcel, see also Eq. 1. The creation and placement of parcels, and hence the cloud shape, in the scene is left to the user to decide, which is not always intuitive. In their method there are no explicit parameters available to tune the cloud coverage.

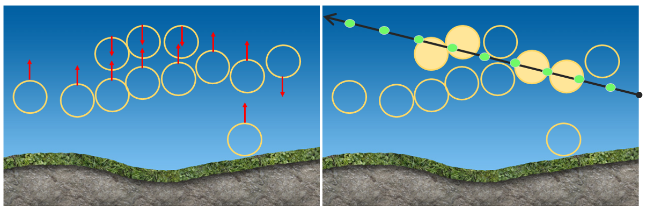


Figure 2: *In the original method, physics is simulated within a bounded domain (octree) at the macro level using parcels (left). In the visualization pass, the parcels intersecting the line of sight are filled with hypertexture based on the physics variables (right). The atmosphere is implicitly represented using temperature and humidity curves.*

We refer the reader to the original article for more in-depth details of the original approach.

3.2. Our model

The original model employs parcels in the 3D as physics primitives, on top of which micro-level procedural amplification using hypertexture is achieved. This imposes certain limitations on the look of the cloud cover generated:

- Parcels are limited to only spherical shape (due to physics) constraining the overall cloud mass to assume somewhat limited appearance
- In order to produce a filled cloud coverage, a few hundreds of parcels are required which makes the visualization component very expensive
- Since the simulation domain is bounded by an octree, cloud coverage is bounded too. This leads to a finite cloud coverage and absence of clouds beyond a certain distance, especially towards the horizon
- The user has to explicitly specify the parcel distribution over the landscape which leads to unequal and somewhat unnatural appearance and spread of clouds

The proposed work eliminates all the above mentioned limitations. The parcel shape or size is irrelevant to the cloud shapes obtained and the method is capable of running at high frame rates even when the cloud coverage in the sky is increased. Since we disentangle the rendering and the physics domain, the cloud coverage has a much larger span, including towards the horizon. Finally, the user does not have to specify the parcel or cloud distribution in the sky.

Fig. 3 demonstrates the approach adopted by our proposed method. In the first step, a single layer of parcels is simulated using the physics from the original method. However, this simulation differs from [GN17] in that the parcels are simulated in a unit octree and contain only a few units comprising a single layer. The parcels are not required to align vertically and may therefore, vary in their altitude. In the second step, the physical attributes from the simulated parcels are mapped and projected on the top of a precomputed noise texture (cloud map) which guides the cloud shape. The purpose of this step is to blend the macro-scale, computationally inexpensive physics from the previous step, with the precomputed cloud-like pattern inherent in this noise texture. Finally, this combined texture is employed to render the physics-inspired clouds within the specified height band y (bounded by h_{min} and h_{max}) in the 3D scene using volume ray casting. The texture could be mapped to a user specified region on the landscape in the $x-z$ direction to cover a certain sky portion (as shown in Fig. 3) or even repeated periodically. It is important to note that the first two steps are largely independent of the scene employed; they depend only on the standard environment parameters like temperature and humidity profile as in the original method.

In the following, we describe the important components of our method in more detail, see also Alg. 1.

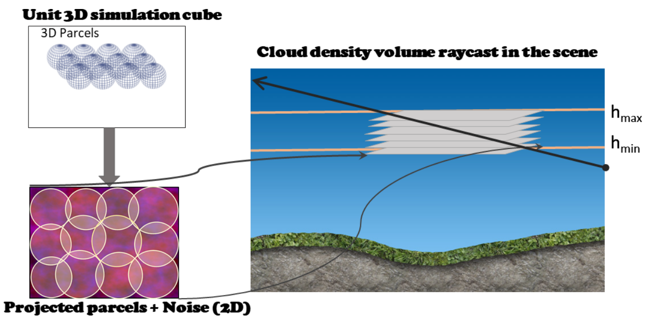

Unit 3D simulation cube
3D Parcels
Cloud density volume raycast in the scene
h_{max}
h_{min}
Projected parcels + Noise (2D)

Figure 3: *In the proposed method, a single layer of a few parcels is simulated in the unit domain (left top). The obtained physics variable values from the parcels are then mapped and projected on top of the cloud map containing cloud generation noise (left bottom). The aggregate texture from the previous step is then mapped within the specified horizontal and vertical range in the 3D scene for volume ray casting (right). Each stacked plane represents a sampling point on the cast ray, for which the same aggregate texture is used.*

3.2.1. Cloud map

In order to improve on the existing method in terms of cloud appearance and performance, our model introduces the concept of 2D cloud map (as used in [Nij18] for static procedural clouds) in conjunction with the aforementioned physics. Cloud map $\mathcal{C}_\mathcal{M}$ is a two-dimensional bounded planar texture consisting of the spatial density values for the cloud mass at a given moment. The idea behind its use is to employ a pregenerated 2D texture of noise that guides cloud coverage and shapes in the sky. Furthermore, instead of the density values, $\mathcal{C}_\mathcal{M}$ stores spatially consistent and continuous noise values which are then used to generate cloud densities. The reason being that its much easier to generate these noise values (for ex, using Perlin noise) than directly obtaining the physically accurate cloud density at various spatial locations. However, this is no limitation and in general a cloud map can be composed of any values that could be translated to the cloud density. These density values are mapped to the sky in along x and z dimensions. The vertical height and span of the cloud is obtained from the atmospheric variables leading to vertical cloud formation. In our case this is sufficient to describe the cloud cover since we assume that the cumulus clouds are single-layered.

The presence of the cloud map gives us a handle to produce realistic cloud shapes but by itself is insufficient to simulate the underlying cloud physics. One possibility could be to use several such maps from successive timestamps to create a continuous animation. In order to simplify the task of storage and animation, we target to achieve our animation with a single chosen $\mathcal{C}_\mathcal{M}$ to evolve cloud cover starting with its birth. In theory, one could evolve it based on a base semi-Lagrangian simulation. However, in practice this idea is confronted with some key impeding issues. Firstly, it would be hard to correlate and blend the density obtained from the grid cells with those in the map texture. In addition to controlling cloud appearance in terms of density, it would also be challenging to maintain the frame-to-frame coherence for animation. Secondly, a detailed semi-Lagrangian simulation would be quite expensive computationally to achieve real-time frame rates. Lastly, this would defeat our basic assumption of employing an efficient model with the disentangled physics guiding visualization. To this end, the hybrid model comprising physics on coarse parcels and visualization at micro-level comes to our advantage, as explained in Sec. 3.2.2.

3.2.2. Physics

In the nature, the cloud layer is often formed due to the temperature boundary layer inversion in the atmosphere. The water vapor continues to rise till it hits the altitude where the temperature inversion takes place. At this point the vapor ceases to lift further up and the condensed cumulus clouds give the appearance of forming a single layer. In their work by [GN17], the parcels composing several layers are created and evolved in the 3D environment. Since we are operating on the cloud map, we need to extend the original formulation for our purpose. Firstly, the intended modification must allow the original physics to operate on $\mathcal{C}_\mathcal{M}$. Secondly, clearly we should be able to optimize both on the physics and rendering fronts computationally since our method works on a single layer of clouds.

In order to work more efficiently on the cloud map, it is mapped with a single initialized layer of parcels covering the entire map or a specific region therein (see also Fig. 3). This connection enables us to correlate the physical attributes from parcels and the shape attributes in $\mathcal{C}_\mathcal{M}$. The parcels are assigned normalized coordinates and dimensions in order to make any 3D to 2D mapping (or vice-versa) easier. Thereafter, after executing the physics on the single layer, we perform a planar projection of the parcels from the unit domain directly on to the image in $\mathcal{C}_\mathcal{M}$ to blend the cloud physics and noise generating texture. Finally, this composite texture is mapped to the user specified 3D space for rendering. The initialized parcels still undergo physics in the unit 3D domain but the assumption of a single layer simplifies the computations, as we explain next.

Each parcel is characterized by a tuple of physical attributes, *<1/0, height, fillDensity>*. These physical values in each simulation loop are projected on a 2D texture $\mathcal{P}$ (which is overlaid on $\mathcal{C}_\mathcal{M}$) by rendering the parcels in an offline frame buffer object texture, storing the tuple with the three values as a *rgb* vector. The first value of tuple for a pixel in the texture $\mathcal{P}$ $(1/0)$ indicates whether it is covered by a cloud parcel or not. Hence no clouds would be formed in the region corresponding to this pixel if it has a value 0. The second value of the tuple gives the normalized height of parcel center whereas the third one stores its filling density. The filling density is obtained using the fraction of water content condensed in a parcel, similar to the original work. The more the condensed fraction, the higher the filling density and the denser the cloud appears. The cloud coverage can also be tuned during noise generation step to incorporate user specified preferences. $\mathcal{P}$ together with $\mathcal{C}_\mathcal{M}$ can then be efficiently used for raycasting the obtained densities (see also Fig. 4), , as explained in Sec. 3.2.3.

Algorithm 1 Cloud animation algorithm

```
 1: Generate cloud map texture $\mathcal{C}_\mathcal{M}$
 2: Generate parcels over $\mathcal{C}_\mathcal{M}$
 3: while (animating) do
 4:     // Physics part
 5:     for all parcel i do
 6:         simulate parcel physics in 3D
 7:     project parcel values in texture $\mathcal{P}$
 8:     // Visualize clouds by volume ray casting
 9:     for each ray and its sampling points do
10:         - determine $\alpha$: visibility and physics parameters from $\mathcal{P}$
11:         - determine $\beta$: corresponding noise parameters from $\mathcal{C}_\mathcal{M}$
12:         - obtain accumulated density value from $\alpha$ and $\beta$
```

3.2.3. Visualization

The next step in each frame after executing cloud physics is to obtain the cloud visualization. In the original work, this is achieved by shooting rays to the screen pixels and accumulating densities for ray parts that hit a parcel. The disadvantage of using this method is that its expensive. Even while taking into account the educated guess of what parcels intersect an octree nodes, the number of intersection tests grow rapidly with the parcels in the scene. It should be noted that in order to cover a significant region in the sky, high parcel counts are required. However, with our tuple values stored in $\mathcal{P}$, we can not only limit the number of parcels simulated but also avoid traversing the regions that are of no interest to us. As mentioned here $\mathcal{C_M}$ is mapped over the landscape for cloud distribution. For each point on the ray traced, we check in the cloud map at that position to generate the corresponding density(noise).

In order to achieve the cloud amplification, we specify a constant vertical cloud span $\Delta h = h_{max} - h_{min}$ in the world scene. This value is obtained from the specified atmospheric conditions in the temperature profile. The height mapped h and the other world coordinates for a particular point on the ray while volume casting is obtained from the tuple in $\mathcal{P}$ (parameter α in Alg. 1) and the corresponding noise values from $\mathcal{C_M}$ (parameter β). Therefore, for each point on the ray being cast, this is the only interesting range to check for the presence of clouds and the related cloud noise generation. In the absence of any noise obtained, we simply render it with the background sky color. On the other hand, if for a given $x - z$ coordinate the map $\mathcal{P}$ contains cloud coverage, procedural noise is generated for the point using $\mathcal{C_M}$. This completely eliminates testing repeatedly for each point traced on a ray if it lies within one of the parcels, which is a very expensive operation. In essence, this operation is replaced by a texture look-up for each point traced on the ray which is much more lightweight both in terms of the memory fetch and computation. As demonstrated in Fig. 3, the aforementioned ray tracing procedure is repeated for each sampled point on the ray (shown as stacked planes), wherein the same $\mathcal{C_M}$ and $\mathcal{P}$ is employed. Additionally, only those screen pixels that do not contain any other geometry have to make this check, since these pixels qualify as the sky pixels where clouds could potentially exist. The fill density corresponding to the proportion of the condensed vapor at each associated point is also present in $\mathcal{P}$ which guides the amplification.

We apply temporal sample anti-aliasing (TSAA) [JGY*11] to the pixels across successive frames to smoothen the appearance of the clouds generated. In the beginning when the clouds just begin to emerge, the visible artifacts arise due to the fact that there are insufficient density values to interpolate from the previous frames. The correspondence between pixels in the current frame and the previous one is obtained by shooting rays on a sphere in the scene. The spherical coordinates used to generate ray direction help us keep track of the pixel correspondences.

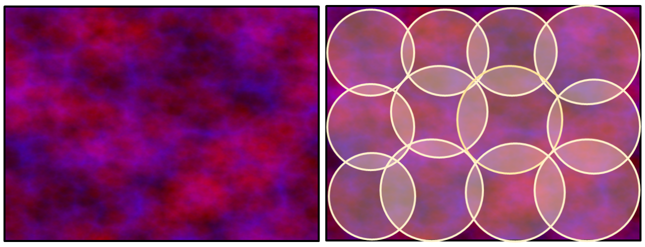


Figure 4: *An example cloud map ($\mathcal{C_M}$) generated using Voronoi tiling in [Nij18] (left). Projection of single-layered parcels containing physics attributes ($\mathcal{P}$) represented as filled circles in 2D on $\mathcal{C_M}$ (right). $\mathcal{C_M}$ guides the cloud shapes whereas $\mathcal{P}$ provides the macro physics attributes for the cloud formation.*

4. Results

The proposed method is implemented and tested using C++ and GLSL on a Windows machine with Nvidia GTX 1080 graphics card. Similar to the original method, we compute physics on the CPU while ray casting is done on the GPU. The screen size is fixed to a resolution of $640X480$ in all the experiments. In addition to the parcels, the background environment is also ray cast using the rendering shaders. It is important to mention here that unlike original method we completely eliminate the use of *uniform buffer object (UBO)* in the fragment shader, which also helps us to optimize the rendering part further. The variables in the UBO are instead now made available through $\mathcal{P}$ texture making it more efficient.

Fig 4 shows an example cloud map $\mathcal{C_M}$ together with the overlay of the projected parcels represented as circles in 2D ($\mathcal{P}$). Here the noise in the cloud map is generated with the Voronoi tiling. Each of the pixels in $\mathcal{P}$ store the aforementioned tuple *<1/0, height, fillDensity>* from the macro-level physics primitives which is passed on to the GPU every frame for the rendering. The overlapping regions between two or more parcels are given the average values.

Fig 5 shows the graphs for a typical parcel undergoing dynamics. In Fig 5a, the virtual potential temperature (Θ^{parcel} in Eq. 1) of the parcel is plotted against that of the surrounding atmosphere (Θ^{atm}). Θ^{atm} first reduces with the altitude and then gradually increases. It is for this reason that the rising parcel loses buoyancy after a while and begins oscillatory motion around the point where its virtual potential temperature graph intersects with that of the atmosphere. This oscillation in the height of the rising parcel as a function of time is also visible in Fig 5b. The fraction of the condensed water present in the parcel is also affected by the change in altitude. In the supporting movie clip showing animation, some fluctuations in cloud development (condensation, height) are noticed due to this oscillatory movement of the parcels. Finally due to the process of entrainment and detrainment, the water content of the parcel reduces with time. This is due to the fact that detrainment rate is higher than the entrainment for most of the lifetime of the parcel. Both of these trends are visible in Fig 5c. Fig 1 and 6 show the cloud development process for two different scenes. The fill density of hypertexture and the parcel height changes in accordance to the condensed water fraction in different parcels (as a function of the underlying physics parameters). The clouds in Fig 6 begin to decay once the cloud development cycle reaches its maturity.

In Fig 7, the proposed method is compared with the original technique with the help of panoramic images from both. It is clear that our method with the cloud map achieves animation leading to a much more realistic coverage and distribution as compared to [GN17] when purely object domain parcels are used. Furthermore, the overall frame rates achieved in Fig 7b is roughly around twice than in Fig 7a. Periodic textural repetition in the $x - z$ plane is allowed to enhance the cloud coverage in the latter case. In order to achieve a similar coverage with the original method, one would need a much larger parcel count. This not only makes the ray casting much more expensive (entails searching and rendering more parcel volume), but also begins to strain the physics part since it is executed on the CPU. Tab 1 gives the frame rates for animation and visualization of varying maximal coverage percentage of clouds (specified by the user). 12 parcels are used in all the cases to generate $\mathcal{P}$ to be overlaid on $\mathcal{C}_{\mathcal{M}}$ and the maximal cloud coverage percentage is tuned by altering filling density of hypertexture as per physics parameters. For 0.1% coverage very little cloud cover exists and this could be also be taken as the approximate frame rates when no clouds are born.

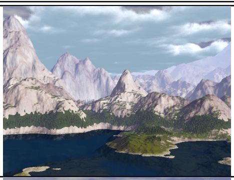
Coverage (in %)	Frame rates	Image
0.91	202	
0.71	192	
0.51	188	
0.1	184	

Table 1: *Overall frame rates of the single-layered cumulus cloud animation and visualization as a function of the maximal cloud coverage which is specified in the approximate percentage of the sky covered with the clouds. Images from the animation for the corresponding coverage are shown in the third column.*

Our experiments suggest that TSAA improves the rendering quality of the clouds significantly, while the efficiency is not affected much. Raytracing background scene (apart from sky) also takes up a part of the computational time, which is variable and depends on the scene complexity. We noticed a boost of around 50-100 fps when switching off the rendering of the background terrain. In Fig. 7, we have chosen the same background for the comparison of the proposed method with the original technique. In [GN17], the frame rate for the simulation and visualization combined drops to 7 fps when 85 parcels are employed. This high count is required in their work to obtain a reasonable cloud coverage. A similar coverage obtained using our method together with the background scene when raycast runs at around 200 fps (see Fig. 1). Our measurements suggest that we spend less than 10% of the total computational time on simulating the parcel physics and projecting them on the cloud map, while the remaining time is spent on the rendering.

5. Conclusions

We have presented an efficient and realistic physics-inspired procedural approach to animate single-layer of cumulus clouds in real-time on the GPU. Modifying the original method, we have projected the parcel physics to 2D (instead of 3D) on top of a cloud map to generate realistically evolving cloud patterns. This significantly reduces the computational overhead of visualization while performing the volume ray casting, in addition to reducing the number of parcels that are needed to undergo physics. Our technique is a perfect fit for cloud animation in the games, flight simulators and other applications where in addition to convincing appearance, real-time performance is a necessity.

One limitation of our method is that in its current form it can only handle single-layered clouds. In future, it could be extended to handle the multi-layered clouds efficiently. Another extension would be to modify the cloud map noise as function of space or time while keeping the same parcels for the physics to support the cloud cover over an infinite landscape. Cloud rendering itself could be improved by incorporating more accurate light interaction with the surrounding environment.

6. Acknowledgements

We thank the anonymous reviewers for their valuable feedback. We would also like to thank Inigo Quilez for the shadertoy scenes *Rainforest* and *Terrain tubes* and Dave Hostins for *Mountains*, which we have employed in our visualization.

References

[BBZ*16] Bi S., Bi S., Zeng X., Lu Y., Zhou H.: 3-dimensional modeling and simulation of the cloud based on cellular automata and particle system. *ISPRS International Journal of Geo-Information 5*, 6 (2016). 2

[BN04] Bouthors A., Neyret F.: Modeling Clouds Shape. In *Eurographics (short papers)* (Grenoble, France, Aug. 2004), Galin E., Alexa M., (Eds.), Eurographics Association, pp. –. URL: https://hal.inria.fr/inria-00537462. 2

[Day05] Day J. A.: *The Book of Clouds*. New York, NY: Sterling Publishing Co, 2005. 2

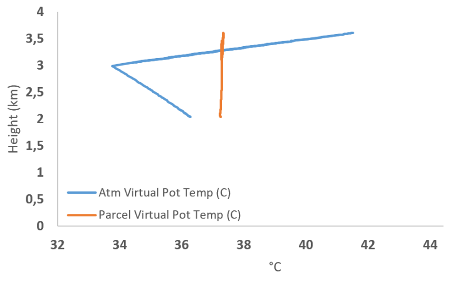
[DG17] Duarte R. P., Gomes A. J.: Real-time simulation of cumulus clouds through skewt/logp diagrams. *Comput. Graph. 67*, C (Oct. 2017), 103–114. URL: https://doi.org/10.1016/j.cag.2017.06.005, doi:10.1016/j.cag.2017.06.005. 2

[DKY*00] Dobashi Y., Kaneda K., Yamashita H., Okita T., Nishita T.: A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (2000), SIGGRAPH, pp. 19–28. 2

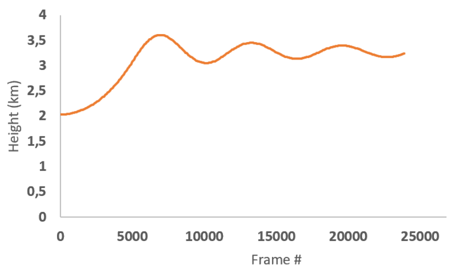
[DYN06] Dobashi Y., Yamamoto T., Nishita T.: A controllable method for animation of earth-scale clouds. In *Proc. of CASA* (2006), pp. 43–52. 2

[FBDY15] Ferreira Barbosa C. W., Dobashi Y., Yamamoto T.: Adaptive cloud simulation using position based fluids. *Computer Animation and Virtual Worlds 26*, 3-4 (2015). 1, 2

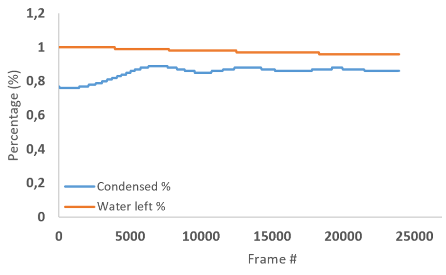
[GD07] Grudziński J., Dębowski A.: Clouds and atmospheric phenomena simulation in real-time 3d graphics. In *Computer Vision/Computer Graphics Collaboration Techniques* (Berlin, Heidelberg, 2007), Gagalowicz A., Philips W., (Eds.), Springer Berlin Heidelberg, pp. 117–127. 2



(a)



(b)



(c)

Figure 5: *Plots for a typical parcel showing (a) variation of the atmospheric vs. parcel virtual potential temperature with the altitude, (b) variation of the parcel height as the simulation progresses, (c) overall remaining and condensed water fraction in the parcel with the simulation progression.*

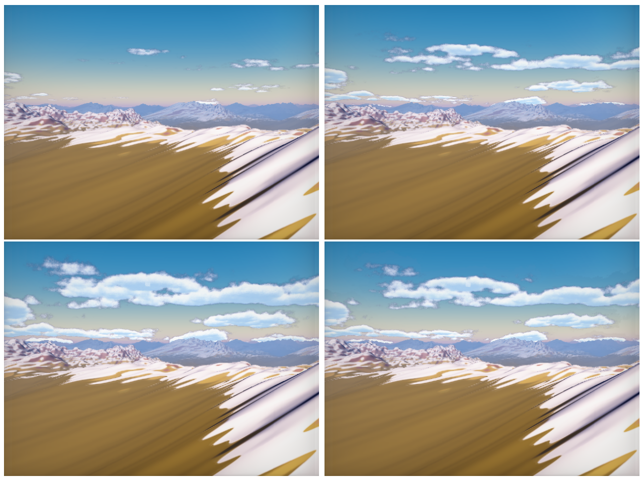


Figure 6: *Timelapse of the cumulus cloud development using the proposed approach. Only 12 parcels are used in the texture $\mathcal{P}$ to generate this cloud cover. After the full development, the clouds begin to decay. Note the distribution of clouds far towards the horizon which is harder to obtain in the original method in [GN17].*

[GN15] Goswami P., Neyret F.: Real-time landscape-size convective clouds simulation. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (2015), I3D, pp. 135–135. 2

[GN17] Goswami P., Neyret F.: Real-time landscape-size convective clouds simulation and rendering. In *VRIPHYS 2017 : 13th Workshop on Virtual Reality Interaction and Physical Simulation, 23-24 April 2017, Lyon, France* (2017), pp. 1–8. 1, 2, 3, 4, 5, 6, 7

[HBSL03] Harris M. J., Baxter W. V., Scheuermann T., Lastra A.: Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), pp. 92–101. 2

[JGY*11] Jimenez J., Gutierrez D., Yang J., Reshetov A., Demoreuille P., Berghoff T., Perthuis C., Yu H., McGuire M., Lottes T., Malan H., Persson E., Andreev D., Sousa T.: Filtering approaches for real-time anti-aliasing. In *ACM SIGGRAPH 2011 Courses* (2011), SIGGRAPH '11, pp. 6:1–6:329. 5

[KMM*17] Kallweit S., Müller T., Mcwilliams B., Gross M., Novák J.: Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Trans. Graph. 36*, 6 (Nov. 2017), 231:1–231:11. 2

[KVH84] Kajiya J. T., Von Herzen B. P.: Ray tracing volume densities. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 165–174. 2

[Mih11] Mihalis L.: *First Principles of Meteorology and Air Pollution*, vol. 19. 2011. doi:10.1007/978-94-007-0162-5. 1

[MM13] Macklin M., Müller M.: Position based fluids. *ACM Trans. Graph. 32*, 4 (July 2013), 104:1–104:12. 2

[MMPZ12] Miller B., Museth K., Penney D., Zafar N. B.: Cloud Modeling And Rendering for "Puss In Boots". In *ACM SIGGRAPH Talk* (2012). 1

[Ney97] Neyret F.: Qualitative simulation of cloud formation and evolution. In *8th Eurographics Workshop on Computer Animation and Simulation (EGCAS)* (sep 1997), pp. 113–124. 2

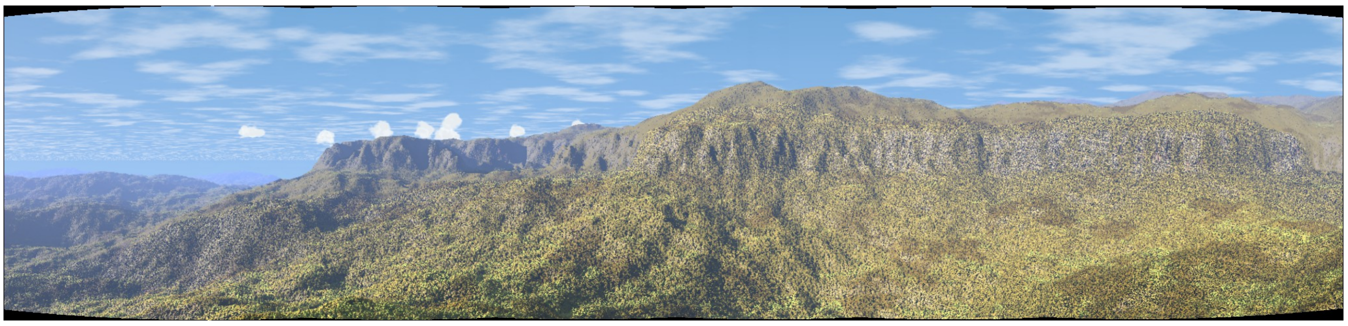
[Nij18] Nijhoff R.: Himalayas, 2018. URL: https://www.shadertoy.com/view/MdGfzh. 4, 5

[SMST] Soares P. M. M., Miranda P. M. A., Siebesma A. P., Teixeira J.: An eddy-diffusivity/mass-flux parametrization for dry and shallow cumulus convection. *Quarterly Journal of the Royal Meteorological Society 130*, 604. 2

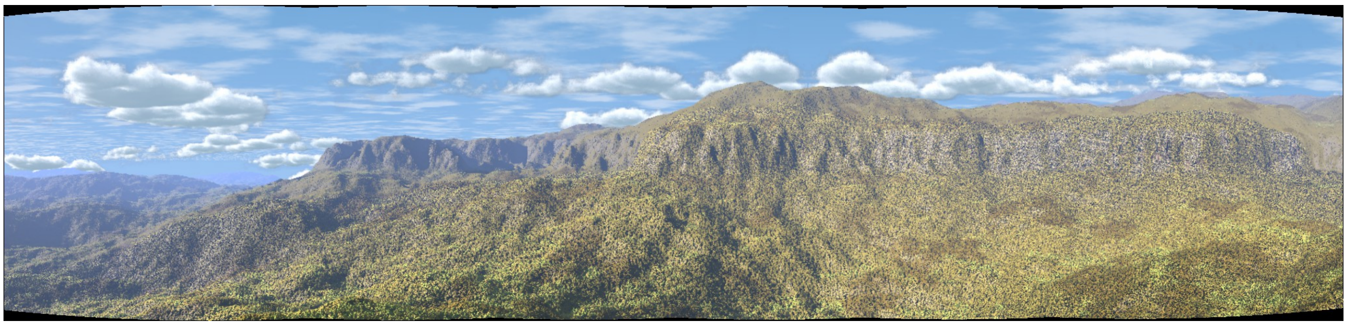
[SSEH03] Schpok J., Simons J., Ebert D. S., Hansen C.: A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), SCA, pp. 160–166. 2

[WCGG18] Webanck A., Cortial Y., Guérin E., Galin E.: Procedural Cloudscapes. *Computer Graphics Forum 37*, 2 (2018). URL: https://hal.archives-ouvertes.fr/hal-01730789. 2

[YW11] Yu C.-M., Wang C.-M.: An effective framework for cloud modeling, rendering, and morphing. *J. Inf. Sci. Eng. 27* (2011), 891–913. 2



(a) *Original method*



(b) *Proposed method*

Figure 7: *Panorma comparison of the cumulus cloud distribution and shape over landscape (textural periodicity enabled) using , (a) original method and (b) proposed method. We employed 12 parcels in each case to generate this cloud cover in both (a) and (b). The proposed animation method is running roughly at two times frame rates when compared to the original method.*