Modeling and Rendering of Volumetric Clouds in Real-Time with Unreal Engine 4

Łukasz Nowak[1(✉)], Artur Bąk[1(✉)], Tomasz Czajkowski[2], and Konrad Wojciechowski[1]

[1] Polish-Japanese Academy of Information Technology, Warsaw, Poland
{lnowak,abak,konradw}@pjwstk.edu.pl
[2] Orka Postproduction Studio, Warsaw, Poland

Abstract. Simulation of realistic clouds is a difficult task that graphic designers have been trying to achieve for many years. Clouds are necessary for video games and animated movies with outdoor scenes.

Unreal Engine 4, currently one of the most advanced game engines, offers various solutions for implementing such effects. In this paper, we present the effective methods for obtaining realistic real-time clouds in Unreal Engine 4 and provide some new extensions to make the result even better. The methods described are suitable for modeling and rendering both a single cloud and the entire sky filled with clouds. Amongst proposed techniques are modeling clouds by using noise functions, modeling by using a simplified simulation of cloud forming phenomena, rendering by using Ray marching technique and using fluid simulation to allow interaction of clouds with other objects on the scene.

Keywords: Clouds · Volume rendering · Unreal Engine 4 · Real-time

1 Introduction

Clouds are a significant part of the sky, which we can observe every day. This is why adding them to our virtual skies noticeably improves their realism. Game developers and animated movie makers, being conscious of this, used various techniques to draw clouds on screen. Creating virtual clouds is not always an easy task and it is still researched, now for over 40 years. The difficulty level of making clouds depends on how important they are in our product. If we want to create a video game whose action is settled only on the ground level, we could limit ourselves to simple tricks which make sky look convincing for the player. But when we are making an aircraft simulator or an animated movie we could be forced to use more advanced methods, especially if achieving real-time operation is necessary.

In this paper, we would like to present the results of our research on obtaining realistic volumetric clouds working in real-time using Epic Games' Unreal

L. J. Chmielewski et al. (Eds.): ICCVG 2018, LNCS 11114, pp. 68–78, 2018.
https://doi.org/10.1007/978-3-030-00692-1_7

Engine 4 game engine [20]. Some of the methods described have already been implemented and used by us in Unreal Engine 4, while others are proposals that we are going to try to implement and test in the future.

Next section presents existing tools and previous work done in field of creation of volumetric clouds. Section 3 describes our approach and is divided into five areas (modeling, rendering, lighting, shadows and interaction with other objects) where we presents methods used or considered by us. Section 3 also contains brief explanation of Unreal Engine 4 terminology and cloud forming process. Section 4 presents current results of our approach and in Sect. 5 future works related to our system can be found.

2 Previous Work

As mentioned earlier, research on generating clouds by computer started many years ago and it is still going on. At the beginning of our research, complete solutions for Unreal Engine 4 were sought. There is one commercial solution for generating sky filled with volumetric clouds and with weather system that has integration with Unreal Engine 4 through editor plugin, Simul's trueSKY [16]. trueSKY may give good visual results and is undoubtedly great solution for creating realistic skies but it does not suit all of our needs. It is proprietary and closed, it cannot be used for creating single cloud, and it does not offer any support for interaction between clouds and other objects on the scene.

Other commercial software that was considered for cloud creation is NVIDIA's Flow [17] from GameWorks suite. Flow allows creating volumetric fire and smoke with fluid simulation built-in in real-time. It has integration with Unreal Engine 4 through custom modifications in engine source code. Unfortunately, again, Flow does not meet all of our expectations as it is non-deterministic and its integration with Unreal Engine 4 has a few flaws concerning lighting and shadows (rendered volume does not cast shadows and other objects on the scene cannot cast shadow on volume). Another disadvantage is that it is a closed source software.

There are also a few hobbyistic projects aiming for volumetric clouds creation presented and discussed mostly on the official Unreal Engine 4 forum [14], two of them are worth mentioning, [18,19]. Both of them are not released for public.

Finally, there is a content plugin for Unreal Engine 4 created by Epic Games' principal technical artist Ryan Brucks, which contains Blueprint scripts and materials implementing ray marching rendering technique and fluid simulation with possibility of user interaction in two dimensions. Plugin is available at [15]. We currently use some solutions included in this plugin and several methods proposed by Brucks, which are described on his personal web page, [11,12]. From now on this plugin will be called *ShaderBits plugin* in the rest of this paper.

Leaving Unreal Engine 4 and focusing on general research on creating clouds, a few works are worth mentioning. First of them is the Master's thesis of Taxén [4], which describes in detail the physical processes related to cloud forming and also presents different types of clouds modeling. A practical example of

using Perlin noise for clouds modeling was presented in [7]. In [2] a proposition of modeling algorithm inspired by clouds forming phenomena was presented. [1] describes clouds modeling method based on [2], which is closer to a simplified simulation of clouds forming phenomena.

In NVIDIA's GPU Gems books, the subject of fluid simulation in real-time was raised. [5,6,9] from respectively first, second and third volume of GPU Gems, describe algorithms for fluid simulation based on Navier-Stokes equations and flow simulation with complex boundaries. Fluid simulation in ShaderBits plugin is actually based on [5].

Another important subject in terms of rendering is lighting. A brief description of light scattering in clouds is in [4]. Approximation of light scattering in clouds was proposed by Harris in [8].

Finally, two works based on released video games are also notable. In [3], we can see an approach for real-time atmospheric effects existing in Crytek's CryEngine 2 [21]. SIGGRAPH 2015 presentation of Guerrilla Games [10] presents a complete pipeline of volumetric clouds creation in their video game including the results of their research and approach for both modelling using noise and rendering by ray marching technique.

3 Overwiev of Methods Used in Our System

This section describes the methods already used in our system and methods that could be implemented and tested by us in Unreal Engine 4.

3.1 Brief Overview of Unreal Engine 4 Terminology

In order to fully understand the solutions proposed and presented in this paper a knowledge of basic terminology existing in Unreal Engine 4 may be required. Unreal Engine 4 is a complex game engine and offers a lot of possibilities. Most important tool in Unreal Engine 4 (UE4 for short) is Editor, a complete tool for game creation. UE4 Editor, which will be also called just *editor* for the rest of this paper, allows creating and setting of maps, materials, Blueprint scripts and many other content.

Map in Unreal Engine 4 is name for scene or level.

UE4's material, which will be called just *material*, will be considered as a pixel shader in this paper. In fact, materials are more complex and can additionally act as vertex or geometry shader. Materials can be created and edited in Material Editor which is a part of Unreal Engine 4 Editor. Material Editor is a graphical tool in which we create material by dragging and dropping blocks, performing various operations (e.g. addition or subtraction), called nodes. We also have possibility to create our own nodes, which are known as Material Functions. Material can be applied on mesh to change its appearance.

Blueprint Visual Scripting is another important part of Unreal Engine 4. It is a complete object-oriented gameplay scripting system. Blueprint scripts can be

created and edited in editor, and similarly to materials they are created visually by dragging and dropping blocks responsible for various operations.

Render Target is a special type of texture on which we can draw in runtime. As an example, we can draw the materials content on this texture. From now on the term *rendering to texture* will stand for drawing material to Render Target in the rest of paper.

Objects on scene in Unreal Engine 4 are called *actors*.

3.2 Brief Description of Cloud Forming Process

Clouds consists of very small water droplets with radius of approximately 50 µm or ice crystals. When floating up in troposphere, the water vapour (which e.g. comes from sea or ocean heated by sun) gets cooled, as the temperature in the atmosphere drops with increase of altitude. When the vapour temperature drops to a certain level called a *dew point* due to condensation or deposition, it is changed to water droplets or ice crystals. Water droplets combined together form a cloud that we could see on the sky. When the water droplet in the cloud grow, it could become too heavy, change into a rain drop and fall from the cloud to the ground.

Most of the clouds we can see on the sky can be divided to 10 basic types based on appearance and altitude. Brief presentation of these 10 types can be found at third section of [1].

3.3 Modeling

Our system for creating volumetric clouds in Unreal Engine 4 consists of two parts, modeling and rendering. Modeling is a process of which the result is cloud (or clouds) density data. Usually, density data is stored in 3D textures, but unfortunately Unreal Engine 4 does not support 3D textures. Known workaround for this limitation is proposed in [11] use of, so called, *pseudo volume textures*. The pseudo volume texture is a regular 2D texture containing slices of volume arranged in a grid. Each slice consists of volume density data at a particular height. An example of such a pseudo volume texture is shown in Fig. 1. The more slices in pseudo volume texture, the more detailed will be rendered volumetric object, but also it will take more time to process the texture in the material.

Although one can create such a texture procedurally in a tool like Houdini and export it to the editor, we focused on methods for generating pseudo volume textures without user interaction and the need for external software. We consider two methods of modeling, using the noise function and running a simulation based on a simplified cloud forming phenomena. Unreal Engine 4 material editor contains a built-in node named *Noise*, which generates the noise for given (on node input) position vector. Noise node has a few parameters possible to tweak where most important of them is *Function*, which allows to select the used noise function. Available functions are *Simplex*, *Gradient*, *Fast Gradient* and *Voronoi*. Each function has its own complexity, which significantly affects the frame rate. The most simple and fastest function is Fast Gradient and the most complex

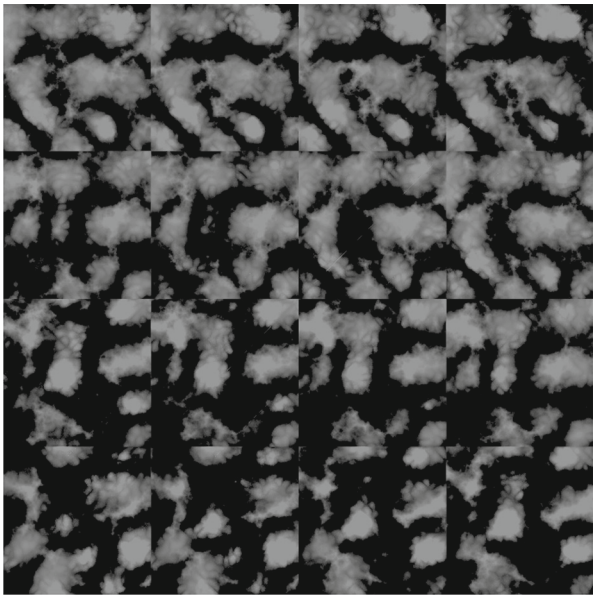

Fig. 1. Example pseudo volume texture with 4×4 grid of slices.

and slowest is Voronoi. Using the Noise node we can create a material that can be used for generation of pseudo volume texture by drawing it to the Render Target. In such material, either only one Noise node can be used or a few Noise nodes with different parameters values can be combined. Mixing several Noise nodes or using complex functions is not possible in real-time if we need to render our material to texture in every frame, e.g. if we want to animate our clouds depending on time. The possible solution can be a one-time generation of the noise texture with expensive functions like Voronoi and passing it to the material drawn every time to Render Target as a parameter. Then the material calculated every frame would contain only the Noise nodes using cheap functions like Fast Gradient. Material generating density texture by noise may require additional operations to make the clouds looking more natural and resemble actual clouds of a given type.

The second method of generating the density pseudo volume texture with clouds is the use of algorithms inspired by the cloud forming phenomena. Performing an exact simulation of all physical phenomena is not possible and would be too complex to calculate in real-time. This is why the algorithms considered by us for implementing in Unreal Engine 4 are very simplified comparing to natural processes. At the moment we have not implemented any of such algorithms yet but in the future we plan to implement the ones described in [1,2]. To achieve a scene with volumetric clouds in real-time using these algorithms, the implementation of them in material (processed on GPU) may be necessary. One possible approach is using the pseudo volume texture to store the simu-

lation grid. Such a texture would be created and populated with initial values in Blueprint script (executed on CPU) and then processed in the material performing the simulation in each next frame. Texture channels can be used to store up to four parameters of grid cell in a single texture.

3.4 Rendering

We use a ray marching method implemented according to [12] for rendering clouds in our system. The Ray marching algorithm is implemented in a separate material. Such material is then applied to the mesh, which was chosen to contain clouds. Method presented in [12] and used by us is suited for cube-shaped meshes. The general idea of the ray marching method consists of "shooting" rays from every pixel of the mesh (with ray marching material applied) and moving through them with constant step defined by the user. At each step, operations defining this pixel colour are executed, e.g. the amount of absorbed light is added up. Number of steps is finite and defined by user. Usually, the end condition for a given pixel is the execution of all number of steps. Graphical presentation of ray marching technique is presented in Fig. 2.

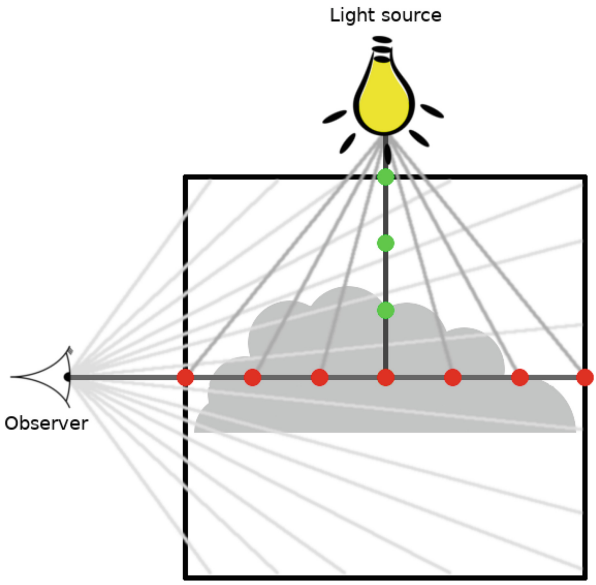

Fig. 2. Brief overview of ray marching technique. At every step (marked by red circles) of ray coming from observer or more precisely mesh' pixel, additional steps (green circles) to light source are performed for volume shading calculation. (Color figure online)

3.5 Lighting

Implementation of a lighting model based strictly on the real world can be difficult and impossible in real-time. This is due to how the light scatters in the cloud. The most difficult task is the approximation of in-scattering existing in the cloud (multiple scattering of light inside the cloud). We currently use the lighting model proposed in [12], which implements only out-scattering. It gives a decent visual result but we consider implementation of more realistic looking solution. Similar approach was presented in [2] and extension of that method in [8].

3.6 Shadows

For obtaining a shadow of the cloud we created a simple material with *Blend mode* (material parameter) set to *Masked*, which takes the pseudo volume texture as a parameter. The material calculates the shape of the cloud based on the pseudo volume texture by summing the density values from its every slice in a given position. Such material is then applied on the regular UE4 plane mesh. The plane mesh is set to not be visible, but casts its shadow. Thanks to the usage of Masked Blend Mode, the shadow shape matches the cloud shape. The shadow shape calculated in this material is in fact a projection of the cloud to the XY plane. One drawback of this method results from this fact, the created cloud shadow will always present the cloud in the XY plane projection independently of the light direction. But for the scenario in which the clouds are high in the atmosphere, this problem would not even be noticeable by the observer in most cases.

The cloud should not only cast shadows, but also allow other objects to cast their shadows on it. Two solutions of this problem were considered by us, using a distance field shadow as in [12] and using a custom per object shadow map presented in [13]. First method gives good visual results but has one drawback, the distance field generation in Unreal Engine 4 does not work with *Skeletal meshes*, i.e. meshes containing the skeleton for animations. We have found a workaround for this limitation by using *sockets*. Sockets allow to connect any mesh to specific Skeletal mesh bone. We connected regular spheres with changed scale and/or rotation to some bones in a way that shape of mesh was covered by these spheres. Spheres were set to be not visible but cast shadow. As added spheres are not Skeletal meshes, distance fields can be generated for them. This way we have obtained an approximated shadow of Skeletal mesh on volumetric cloud. This method allows the shadow to move as the added spheres are moving along with the bones to which they are connected while the Skeletal mesh is playing animation. One inconvenience of this workaround is that additional spheres must be added and connected to bones manually in Unreal Engine 4 editor. The second method of achieving the shadows of other objects on clouds, i.e. using custom per object shadow map, was implemented in UE4 according to [13]. Although this method does not have a problem with various types of meshes (e.g. Skeletal

meshes) it has other significant drawbacks. The first drawback is a poor performance compared to the previous method. The second drawback is the necessity of making the shadows softer to look more realistic on cloud that may impact the performance even more. Another problem are graphical artifacts like distortion or disappearance of shadow when object casting it is moved away from the cloud (as it could not be captured by *Scene Capture* component). Because of these drawbacks we decided to use the first of described methods, i.e. distance fields shadows.

3.7 Interaction with Other Objects on Scene

In most of video games, the player does not interact directly with the clouds and even in most aircraft simulators, flying through the clouds does not change their appearance. However such feature could be desired e.g. in animated movie. This is why we used a fluid simulation for allowing other objects on the scene to reshape cloud by e.g. flying through it. The ShaderBits plugin contains the fluid simulation based on [5] implemented in materials, where each stage of simulation is implemented in a separate material. Unfortunately, by default, the interaction works only for 2 dimensions. We have modified it to work in 3 dimensions with volumetric objects. As such simulation is computationally expensive, we use it only with the single cloud at this moment. Figure 3 presents an example effect of fluid simulation in cloud.

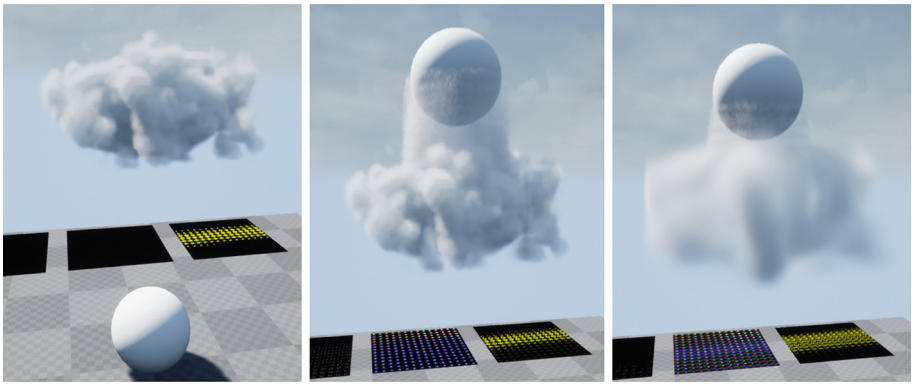

Fig. 3. Fluid simulation in cloud. Left: before sphere moved into cloud
Middle: after sphere moved through cloud
Right: after few passes of sphere through cloud

4 Results

The results of our work are slightly different for creation of one cloud and multiple clouds, which can fill the sky. Our setup for single cloud consists of an actor

Blueprint script and materials for ray marching, fluid simulation stages and cloud shadow. Appropriate parameters in Blueprint script allow to set the ray marching material, the cloud density pseudo volume texture or material for generating one, the pseudo volume texture dimensions and various settings for fluid simulation. Interaction with other objects on the scene is limited to only one object at this time. Figure 3 presents the single cloud with fluid simulation.

For multiple clouds, which can fill the sky, we have very similar setup. Again, it consists of the actor Blueprint script and materials for ray marching, pseudo volume texture generation and clouds shadow. As mentioned in subsection dedicated to modeling, we currently use the noise functions for generating pseudo volume texture. We do not use the fluid simulation for multiple clouds setup. In Fig. 4 the sample sky created by our system using Fast Gradient noise function is presented.

Fig. 4. Sky filled with volumetric clouds modeled by Fast Gradient noise function. Images taken from various viewing angles and time.

All of presented results were achieved on Unreal Engine 4.19.2 in 1920×1080 pixels resolution running on desktop computer with Intel Core i7-5820K CPU, 64 GB of RAM and NVIDIA GeForce GTX 970 video card. The frame rate of performed simulations resolves around 25 frames per second.

5 Future Work

Although we managed to generate clouds, our system is still incomplete and further research is necessary. First, our cloud modeling by noise needs to be improved for achieving clouds more resembling existing in nature clouds of various types. In terms of cloud modeling we are also planning implementation of methods in [1,2]. For rendering, the ray marching technique will be used but improvements in lighting area, especially approximation of multiple scattering will be researched. We also consider enhancements in the fluid simulation methods used by us to allow more complex simulation boundaries and support more than one object at time.

Acknowledgements. This work has been supported by the National Centre for Research and Development, Poland, in the frame of Operational Programme Smart Growth 2014–2020 within Sectoral Programme GAMEINN, POIR.01.02.00-00-0115/16 "Innovative use of computer game engine in order to reduce costs and time of production of animated film".

References

1. Miyazaki R., Yoshida S., Dobashi Y., Nishita T.: A method for modeling clouds based on atmospheric fluid dynamics (2001)
2. Dobashi Y., Kaneda K., Yamashita H., Okita T., Nishita T.: A simple, efficient method for realistic animation of clouds (2000)
3. Wenzel C.: Advanced real-time rendering in 3D graphics and games, chapt. 6, Real-time atmospheric effects in games (2006)
4. Taxén, G.: Cloud Modeling for Computer Graphics (1999)
5. Harris, M., Fernando, R. (eds.): GPU Gems, chapt. 38, Fast Fluid Dynamics Simulation on the GPU. Addison-Wesley Professional (2004). ISBN 978-0321228321
6. Li, W., Fan, Z., Wei, X., Kaufman, A., Pharr, M. (eds.): GPU Gems 2, chap. 47, Flow Simulation with Complex Boundaries. Addison-Wesley Professional (2005). ISBN: 978-0321335593
7. Man P.: Generating and real-time rendering of cloud (2006)
8. Harris M.: Real-time cloud rendering for games (2002)
9. Crane, K., Llamas, I., Tariq, S., Nguyen, H. (eds.): GPU Gems 3, chap. 30, Real-Time Simulation and Rendering of 3D Fluids. Addison-Wesley Professional (2007). ISBN 978-0321515261
10. Schneider A., Vos N.: The real-time volumetric cloudscapes of Horizon - Zero Dawn (2015)
11. Brucks R.: Authoring pseudo volume textures. http://shaderbits.com/blog/authoring-pseudo-volume-textures (2016). Accessed 17 July 2018
12. Brucks R.: Creating a volumetric ray marcher. http://shaderbits.com/blog/creating-volumetric-ray-marcher (2016). Accessed 17 July 2018
13. Brucks R.: Custom per object shadowmaps using blueprints. http://shaderbits.com/blog/custom-per-object-shadowmaps-using-blueprints (2016). Accessed 17 July 2018
14. Unreal Engine 4 Forums. https://forums.unrealengine.com/. Accessed 17 July 2018

15. Training Livestream - Realtime Simulation and Volume Modelling Experiments (Unreal Engine 4 forum thread). https://forums.unrealengine.com/unreal-engine/events/116634-training-livestream-realtime-simulation-and-volume-modelling-experiments-may-2-live-from-epic. Accessed 17 July 2018
16. Simul trueSKY Web Page. https://simul.co/truesky/. Accessed 17 July 2018
17. NVIDIA Flow Web Page. https://developer.nvidia.com/nvidia-flow. Accessed 17 July 2018
18. Volumetric cloud system overview (Unreal Engine 4 forum thread). https://forums.unrealengine.com/community/community-content-tools-and-tutorials/32639-volumetric-cloud-system-overview. Accessed 17 July 2018
19. Working on Volumetric Clouds in UE4. https://80.lv/articles/working-on-volumetric-clouds-in-ue4/. Accessed 17 July 2018
20. Unreal Engine 4 Web Page. https://www.unrealengine.com/. Accessed 17 July 2018
21. CryEngine 2 Web Page. http://www.crytek.com/cryengine/cryengine2/overview. Accessed 17 July 2018