Volumetric Real-Time Smoke and Fog Effects in the Unity Game Engine

A Technical Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Jeffrey Wang

May 6, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Jeffrey Wang

Technical advisor: Luther Tychonievich, Department of Computer Science

Volumetric Real-Time Smoke and Fog Effects in the Unity Game Engine

Abstract

Real-time smoke and fog volumetric effects were created in the Unity game engine by combining volumetric lighting systems and GPU particle systems. A variety of visual effects were created to demonstrate the features of these effects, which include light scattering, absorption, high particle count, and performant collision detection. The project was implemented by modifying the High Definition Render Pipeline and Visual Effect Graph packages for Unity.

1. Introduction

Digital media is constantly becoming more immersive, and our simulated depictions of reality are constantly becoming more realistic. This is thanks, in large part, due to advances in computer graphics. Artists are constantly searching for ways to improve the complexity of their effects, depict more realistic phenomena, and impress their audiences, and they do so by improving the quality and speed of rendering – the algorithms that computers use to transform data into images (Jensen et al. 2010).

There are two breeds of rendering: real-time and offline. Offline renders are used for movies and other video media. The rendering is done in advance by the computer, saved as images, and replayed later as a video to the audience. Conversely, real-time rendering aims to produce images on the fly and display it to the user as fast as possible. This makes it suited to interactive applications, like user interfaces, artist tools, visualization software, and most commonly, video games (Teo 2010). Most advances in real-time graphics come from the video game industry (Reina, 2020).

These two kinds of renders aim to fit different needs, so their approaches change to match. The time allowed for an offline render is only limited by the patience of the creator; it can take hours to produce a single image, or frame. Conversely, many gamers demand that their machines render 60 frames for optimal interactivity. Because of this performance target, the algorithms used for real-time rendering can look very different from offline graphics (Jensen et al., 2010).

A common type of visual effect in games and movies are volumetric effects. These effects include things any kind of particulate matter in the air, like fog, smoke, steam, and dust clouds. These effects can be approximated using billboard particles – large, semi-transparent 2D images that represent the effect as an image. (Unity, 2020) However, if the artist wants to depict the effect from many angles, or in variable light conditions, fog and smoke must be modelled volumetrically in 3D space and lit using appropriate lighting algorithms. Volumetric fog can be rendered efficiently in real-time (Hilaire 2015), but billboards continue to be the most common method of rendering dynamic effects like smoke (Unity, 2020).

With recent advances in GPU particle simulations, millions of particles can be simulated in real time on modern hardware. These high-density simulations provide an opportunity to improve the quality of real-time smoke effects by leveraging volumetric lighting techniques, and to bridge the gap between real-time and offline smoke simulations.

2. Related Works

Offline smoke simulations are often generated by running a fluid simulation on density voxels (a technique where space is divided into a 3D grid of cells), but these simulations are costly to run in real-time (Huang 2015). This project uses GPU particle simulations in place of

fluid simulations, using noise to simulate advection phenomena, in order to meet real-time performance criteria.

The Frostbite game engine developed a system for creating volumetric fog effects – the simulation of light scattering and absorption effects through particulate matter in the air. It also developed a particle volumetric shadow system, which allows volumetric fogs and particles to cast shadows on each other and the environment (Hillaire 2015). This project does not directly refer to the approach used by the Frostbite engine, but it is possible that the volumetric fog system in the Unity High Definition Render Pipeline (a package used by this project) references Frostbite's approach. This project focuses on extending these volumetric rendering techniques to complex particle effects.

The Flax game engine features local fog via volumetric fog particle systems in its 1.1 update (Figat, 2021). Users can use the node-based particle editor to generate dynamic local fog effects. Whereas Flax uses particles over a large area to simulate local fog effects, this project focuses on providing dense smoke effects with a high particle count and preserving the artist-friendly tools already present in the Unity game engine.

3. Background

To simplify the game creation process, game engines provide a toolset for creating and rendering games with artist and programmer-friendly tools. These game engines are often paired with other tools for creative professionals, like image editors, 3D modeling tools, and visual effect design tools. These third-party tools are used to create assets, while the assembly of the scene and programming of behavior takes place within the game engine. Unity is a game engine used by this project.

Graphics Processing Units (GPUs) are processors designed to perform many tasks in parallel very quickly. Their main purpose is to allow a computer to render graphics very quickly. Programs that run on the GPU are called shaders. There are four common types of shaders: vertex shaders, which run per-vertex and apply transformations on the object; fragment or pixel shaders, which run for every pixel on screen and calculate the color based on inputs from the vertex shader; geometry shaders, which run once per primitive (usually triangle) and can output new primitives; and compute shaders, which are ideal for generic computations outside of the traditional rendering pipeline (Rodríguez, 2013).

When determining how the project should be implemented, there were two systems which would greatly reduce the work needed for a proof of concept: a system for creating GPU particle simulations, and a system that can create volumetric lighting effects. The Unity game engine meets both criteria with two packages: High Definition Render Pipeline (HDRP) for volumetric lighting, and Visual Effect Graph for GPU simulated particle effects. Both packages have publicly available source code, which was modified for the needs of this project. The implementation was done in Unity 2020.1.17f1, on top of the HDRP version 8.2.0.

3.1 Particle Systems in Real-time Graphics

Smoke visual effects can be powered by a fluid simulation. While real-time fluid simulations are still rarely used, high volumes of particles are commonly employed to create dense visual effects. This is possible through a technique presented by Gareth Thomas at AMD and Microsoft Developer Day in 2014. Thomas's approach to particle simulations is to use the GPU's parallel processing power to update the particle simulation. He also describes how collisions can be handled on the GPU using the depth buffer information in a normal rendering

pass. These techniques allow for performant particle simulations, which are necessary for the visual effects described in this paper.

Unity's Visual Effect Graph feature implements all of these features; it can run particle counts higher than one million on modern hardware, with collisions through the depth buffer. This was tested on a mid-range home PC running on a Nvidia GeForce RTX 2060 graphics card (a mid-range GPU). Note that the naming conventions of Unity abbreviate Visual Effect as VFX to name functions and data used by this package.

3.2 Volumetric Effects in Unity's HDRP

In Unity, density volumes can be created as objects in the scene. The HDRP renders these as a uniform volumetric fog throughout their boundary area and simulates scattering effects from light sources in the scene and extinction from the fog's absorption of light. Through analysis of the HDRP source code, it appears that it uses a similar approach to the Frostbite game engine (Hillaire 2015). They both feature clip space volumes, encode the same data into their voxel buffers, and use temporal data to increase their accuracy. Unlike the Frostbite engine, the HDRP does not support volumetric shadows, which negatively impacts the realism of dense effects like smoke.

The HDRP runs two important volumetric tasks on the GPU: the VolumeVoxelization task, and the VolumetricLighting task. The VolumeVoxelization task takes a list of density volumes in the scene and fills a voxel density buffer. Each voxel in this buffer corresponds to a point in clip space, with a logarithmically scaling depth. It encodes 4 float values: three represent the red, green, and blue scattering, and the last represents the extinction of the fog at that point. The VolumetricLighting task is later in the render sequence. It reads the density buffer created during the VolumeVoxelization step and a list of light sources to calculate the radiance and extinction of the illuminated fog at each voxel. This data is finally used by the sky and fog render step to create the final colors visible by the camera.

4. Modifications to the HDRP

In order to add fog and smoke from particle effects, the particle effects can write to a scattering/extinction buffer within their pixel shader. This buffer can then be used by the render pipeline alongside the existing volumetric density information, so that the fog effects written by the visual effects are incorporated into the VolumetricLighting pass. The new buffer was named VBufferVFXDensity, and it is declared, instantiated, and destroyed alongside the existing VBufferDensity buffer written by the VolumeVoxelization task and read by the VolumetricLighting task. The compute shader for the VolumetricLighting task was updated to read the original density buffer and the new VFX density buffer, adding their contributions together with a linear scaling function.

To make this new buffer visible from every pixel shader, it was made as an Unordered Access View (UAV). UAVs were introduced in Direct3D 11.3 and allows temporally unordered read/write access from multiple threads to a single buffer. HDRP uses several UAVs of this kind, clearing active ones after various stages in the render pipeline. In the interest of time, the API calls to clear active UAVs were removed so that the VFX density buffer would be visible for shaders in all render steps.

5. Working with the VFX Density buffer

The new VFX density buffer can be written to from any shader. The particles' fragment shaders were chosen to handle the writes to the buffer, because these shaders already handle the visualization of particles. Within these shaders, is relatively simple to translate the visualization to the VFX density buffer. Existing tools to control particle size, color, and textures would retain their functionality for fog particles.

5.1 Writing the Test Shader

A custom vertex and fragment shader was written for testing. The vertex shader calculates view space coordinates via the UnityObjectToViewPos pre-included function and includes it in its output. The fragment shader uses the view space coordinates. It also calculates the fragment's logarithmic depth coordinate; the technique and variables needed to do so are referenced from the function EncodeLogarithmicDepthGeneralized within the Common.hlsl file in the core render pipeline package. With all three coordinates, white scattering is added to the VFX density buffer at the location of the fragment. The result shows up as a light cube; there are small banding artifacts because the depth coordinate is discretized and subtle grid artifacts because the voxel grid is lower resolution than the render resolution.

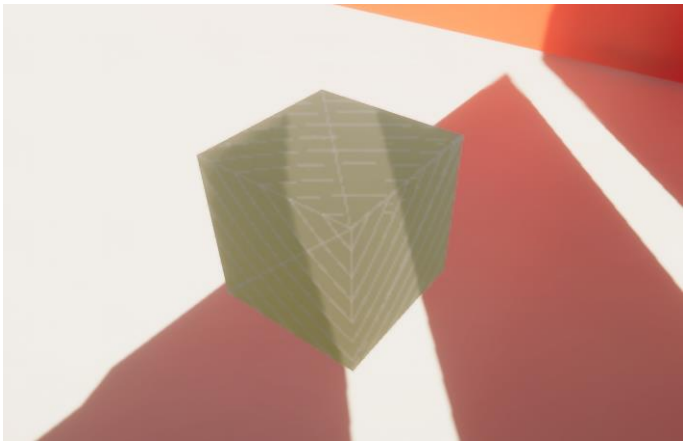

Figure 1. Test fragment shader which outputs fog into the nearest VFX density voxel.

5.2 Visual Effect Graph Modifications

With a working fragment shader finished, results needed to be integrated into Visual Effect Graph. Visual Effect Graph uses template files to generate shaders from its node-based editor system. By editing the shader template files for ParticlePlanarPrimitives within the Visual Effect Graph package, the rendering of visual effects with planar outputs can be changed. The PassVS file was edited to include view space coordinates. The coordinates are calculated with the TransformPositionVFXToView pre-included function. The PassForward file was edited to write to the VFX density buffer using the same technique as the test shader. The data written to the density buffer is equal to $5*[r,g,b,1]*a$, where r, g, b, and a are respectively the red, green, blue, and alpha of the fragment. Afterwards, the fragment's alpha is overridden to 0 so that only their fog contribution is visible. After these changes were implemented, particle effects created with Visual Effect Graph now create fog effects:

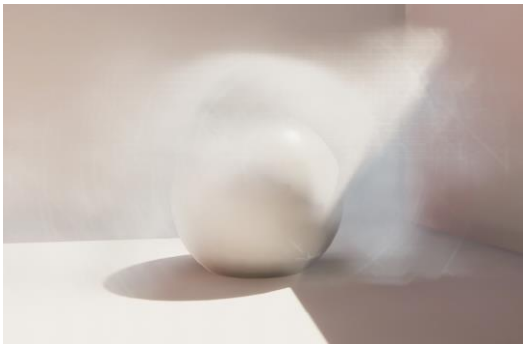

Figure 2. A mist effect created with Visual Effect Graph.

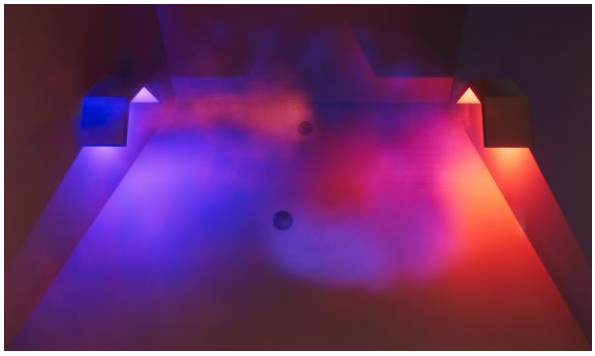

Figure 3. A colored smoke effect that demonstrates depth-buffer collisions.

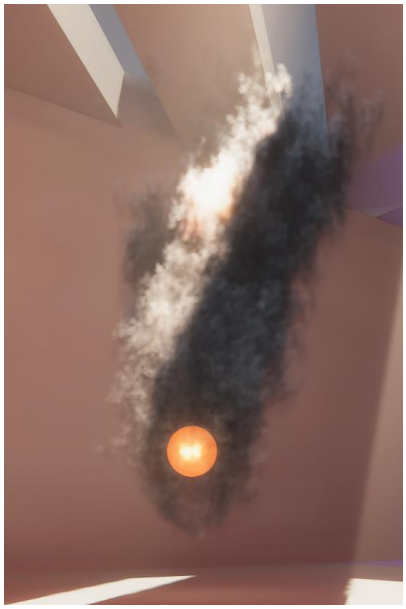

Figure 4. A smoke effect with high particle count and scattering effects.

The results are promising. Figures 2 through 4 demonstrate various effects created the modified HDRP and Visual Effect Graph. The effects demonstrate scattering, collide with the depth buffer, and support high particle counts. However, they do not self-shadow, there are no fluid dynamics in the particle simulation, and they run at a low resolution. In addition, performance takes a significant hit. This may be because of the concurrent asynchronous read/writes to the VFX density buffer. Particles still exhibit both kinds of line artifacts visible on

the test shader. Additionally, the temporal reprojection feature in HDRP which smooths out volumetric lighting now affects performance significantly and causes dark ghosting artifacts on fast-moving particles. Despite these minor issues, the effects are a significant visual improvement over traditionally rendered particles in Unity for smoke and fog.

6. Future Work

The modified HDRP and Visual Effect Graph successfully provided a way of rendering volumetric smoke and fog effects in real-time. The system is effective as a proof-of-concept but needs additional work to be used in most products. For one, the system overrides the default rendering behavior of all particles created with Visual Effect Graph. A more robust system would add a new type of output in the Unity editor and use the templates to enable volumetric fog output only when that output is selected. Future work may also alleviate performance issues and grid artifacts by making read/writes to the VFX density buffer part of a low-resolution render pass. Line artifacts caused by depth discretization may be addressed by blending contributions to the two nearest voxel grid cells on the z-axis. Further research should be done to find a more efficient method of writing volumetric information than a global UAV, as concurrent accesses from multiple threads may impact performance.

Volumetric shadowing would provide a large boost to the realism of effects. With it, dense volumetric VFX would have more realistic lighting and may also cast shadows on the environment. Implementation of this would be a relatively large project. Frostbite's approach to volumetric shadows may provide guidance on this topic (Hillaire, 2015).

The objective of this project, which was to improve the quality of real-time smoke effects using high density particle effects and volumetric rendering techniques, was met by the project. The results obtained are a proof of concept. More efforts need to be taken to eliminate artifacts, improve performance, and improve usability, before this can become a useful tool in most projects.

References

Fiadotau M. (2018) Indie game. In: Lee N. (eds) Encyclopedia of Computer Graphics and Games. Springer, Cham. https://doi-org.proxy01.its.virginia.edu/10.1007/978-3-319-08234-9_188-1

Figat, W. (2021, March 28). Flax 1.1 Released [web log]. https://flaxengine.com/blog/flax-1-1-released/

Figat, W. (2021). Volumetric fog. Flax documentation. https://docs.flaxengine.com/manual/graphics/fog-effects/volumetric-fog.html

Hillaire, S. (2015). Toward unified and physically-based volumetric lighting in *Frostbite*. Presentation at SIGGRAPH 2015, Los Angeles, CA. https://www.ea.com/frostbite/news/physically-based-unified-volumetric-rendering-in-frostbite

Huang, Z., Gong, G., & Han, L. (2015, September 15). Physically-based smoke simulation for computer graphics: a survey. *Multimedia Tools & Applications, 74*(18), 7569 - 7594.

Jensen, H., & Akenine-Möller, T. (2010). The race for real-time photorealism: The coevolution of algorithms and hardware is bringing us closer to interactive computer graphics indistinguishable from reality. *American Scientist, 98*(2), 132-139. http://www.jstor.org/stable/27859478

Reina, G., Childs, H., Matković, K., Bühler, K., Waldner, M., Pugmire, D., ... Krone, M. (2020, April 1). The moving target of visualization software for an increasingly complex world. *Computers & Graphics*, 87, 12 - 29.

Rodríguez, J. (2013, December) The Graphics Rendering Pipeline. *GLSL Essentials*. Packt Publishing Limited. https://subscription.packtpub.com/book/hardware_and_creative/9781849698009/1/ch01lvl1sec10/types-of-shaders

Teo, L. (2010, November 15). CG science for artists – Part 1: Real-time and offline rendering. CG Channel. http://www.cgchannel.com/2010/10/cg-science-for-artists-part-1-real-time-and-offline-rendering/

Thomas, G. (2014, June 2). Holy smoke! Faster particle rendering using Direct Compute [Web log post]. AMD Developer Central. https://www.slideshare.net/DevCentralAMD/holy-smoke-faster-particle-rendering-using-direct-compute-by-gareth-thomas

Unity, Inc. (2020, August 31). Creating fire, smoke & mist effects with VFX Graph in Unity! (Tutorial). https://youtu.be/OCzGXcdyqnQ