HoloLens 2 Performance Characteristics in Point Cloud Rendering

Thy Do, Jinhan Hu, Robert LiKamWa
Meteor Studio, Arizona State University
Tempe, Arizona, USA
thydo,jinhanhu,likamwa@asu.edu

Abstract

The Microsoft HoloLens 2 provides users an interactive medium to 3D models. In this paper, point clouds data rendering is examined. A small model already contain thousands of data points. However, with a device of limited computing capability, HoloLens targeted applications must be efficient with the limited processing capabilities of the device. Furthermore, the applications must be able to render the data smoothly. Using Unity Game Engine, applications are deployed to observe and characterize the supported render pipelines: Built-in and Universal. Five rendering methods are implemented on each of the pipeline: Game Objects, Mesh Topology, Particles System, Compute Shader, and VFX Graph.

Collected results showed the Built-in pipeline is most suitable, while the Mesh Topology is compatible with both pipelines, has the highest GPU and CPU FPS and the lowest batches count. VFX is also a good contender, albeit only the Universal Render Pipeline supports it. This paper provides a basic ground work for further optimization research for the HoloLens 2 and other MR devices using Unity.

Keywords: point clouds data, mixed reality, Unity, HoloLens 2

ACM Reference Format:
Thy Do, Jinhan Hu, Robert LiKamWa. 2021. HoloLens 2 Performance Characteristics in Point Cloud Rendering. In *NSF Computational Imaging and Mixed-Reality for Visual Media Creation and Visualization REU Program (Visual Media REU).* ACM, New York, NY, USA, 9 pages.

1 Introduction

The Microsoft HoloLens 2, is a Mixed Reality (MR) device. A head-mounted display that allows an immersive experience to its users by incorporating gaze tracking, hand gestures, and voice command.

One of the quintessential function of a MR device like the HoloLens 2 is data visualization. This paper examines the visualization of point clouds data, a model comprises of thousands of vertices in the form of xyz vectors. From which, the vectors are mapped onto the device's perceived "real world" matrix in order to overlay the model onto the user's environment [3]. However, the constant usage of sensors, spatial computation and rendering data points can generate performance overhead. For this research, Unity Game Engine is used.

Microsoft's Mixed Reality Took Kit (MRTK) allows developers to deploy applications onto the HoloLens 2. There are many ways to render data points with Unity, which compels the question, which technique offers the best experience? It should offer highest image quality with the most stable framerate. Without these qualities, user may experience eye strain, motion sickness among other side effects. Which may lead to decrease in productivity and can limit the effectiveness of the HoloLens. Yet there are few documentations to answer this question. There are existing solutions that optimize data visualization [4]. The simplest solution is to use Azure Holo App Remoting, but network can be unstable, or unavailable at times. It is better to have a self contained rendering capability.

Unity supports two render pipelines for HoloLens 2: Standard and Universal. The Universal Render Pipeline is intended for mobile devices. Yet the general consensus on which is the better pipeline is case by case evaluation. This paper aims to characterize the performance of HoloLens 2 using several basic rendering techniques; as well as examining those techniques across the render pipelines.

2 Related Works

2.1 Rendering Point Clouds with Compute Shader

Graphic Processors are viable solutions to rasterizing large amounts of data. However, to render large point clouds requires time consuming "hierarchical level-of-detail (LOD) structures" to load and render a fraction of the model at any given time [4]. A basic compute shader encodes the depth and color, in which only the lowest depth points are selected and rendered. Other optimization approaches are applied and evaluated. The research also examined the impact of different vertex ordering algorithms on performance. Peak

Visual Media REU, August 2021, Tempe, AZ, USA

performance was able to render 796 million points at 62 to 64 FPS on an RTX 3090 without using LOD structures [4].

2.2 Remote Rendering

Investigating the bottlenecks of hybrid rendering, visualizing data packets are sent over Wi-Fi network using TCP protocols. The tested devices are the HoloLens and NVIDIA Shield Tablet K1. Without networking, HoloLens can render up to 1000 particles at 30 FPS, while the Shield Tablet can render up to 2500 particles. However, despite the noticeable networking overhead, the major bottleneck factor is the rendering overhead.

2.3 HoloLens 2 Research Mode

The HoloLens 2 comes with an array of sensors, an incredible tool kit for capturing image sensor streams. Research Mode is an open sourced framework for computer vision research by providing functionalities enabling access to raw sensor data [15]. These tools allow the device to capture depth and color, which is useful for object scanning and reconstruction.

3 Background

3.1 Hololens 2

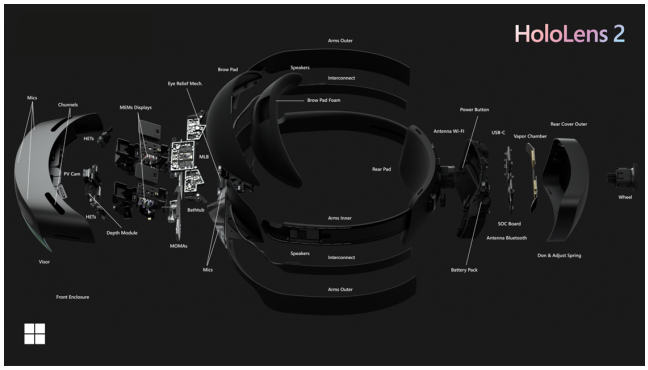


Figure 1. Hololens 2: Individual hardware components that make up the device [5]

The Hololens 2 device arrives equipped with various sensors for head and gaze tracking, depth sensors and microphone for voice command support (Figure1). The vision data is process with a second generation custom-built Holographic Processing Unit (HPU 2.0) [15], located at the front of the device (Figure1). Most importantly, the Hololens 2 has improved computational hardware, a Qualcomm Snapdragon 850 [6], located in the rear (Figure1). An improvement from its predecessor, with 64 bit architecture and 4-GB RAM, allowing better performance.

3.2 Rendering Pipelines with Unity

According to Unity, a rendering pipeline is a series of operations that populate contents of a Unity Scene to the viewing screen [13]. There are 3 render pipelines from Unity: Built-in (Standard) Render Pipeline, Universal Render Pipeline (URP) and High Definition Render Pipeline (HDRP). However, aside from the Built-in Pipeline, only the URP is supported for HoloLens devices [9], and HDRP is not [10].
To set the scene, a scriptable render pipeline allows developers to directly write C# scripts that schedule rendering commands [14]. The URP supports scripting as described.

3.3 Rendering Techniques

3.3.1 Game Objects. Perhaps the most naive approach, rendering each points with individual Unity Game Object. In this case, simple 3D cubes are used, each containing a Transform, Mesh Filter and Mesh Renderer component. A cube is cloned for every vertex and placed at its vector coordinate.

3.3.2 Mesh Topology. The standard method of point cloud rendering, each vertex is read into the Mesh Topology Scripting API. The coordinates of the vertices are then transferred to an array in the Mesh Topology. Each index is rendered as a point, forming the bunny. This method utilizes a single Game Object, with a Mesh Filter and a Mesh Renderer component.

3.3.3 Particle System. Capable of rendering thousands of small meshes, Particle System is efficient at rendering large number of individual points at once [11]. Vertices are applied onto the system, each vertex is a 2D particle set in a fixed position, resulting in a 3D rendered bunny.

3.3.4 Compute Shader. A program that utilizes the graphics processor's capability in parallel computing: fast, multithreaded computation. This technique is best at performing expensive visual effects with millions of particles. The vertex data is read into a compute buffer, then the points are drawn using the Graphic APIs [7].

3.3.5 Visual Effects Graph (VFX). A robust tool for creating complex particle systems, VFX can render million points, leveraging the GPU's computing power. However, VFX requires a Scriptable Render Pipeline, which only URP can support in the HoloLens.

3.4 Profiling Tools

In order to characterize Unity application performance on the HoloLens 2, certain metrics, such as CPU and GPU usage, draw calls, memory and power consumption are needed to be measured and recorded. Such tools are known as profiling tools. Used in this paper are: Unity Profiler and Windows Device Portal (WDP).
There are 14 default profiling profiles packaged with the Unity Profiler, allowing a wide range of characterization. The Unity Profiler can connect to a remote device via local network, enables profiling application on the target platform [12]. Developers can record the playing application and determine the impact of variables such as scripts, assets, project and render settings on performance.

On the other hand, Windows Device Portal provides real time device information from the HoloLens like CPU, I/O, Power Utilization, Memory Usage, etc... [2]; as well as high level device status: battery percentage and temperature. WDP also provides current running processes and their CPU consumption. Most importantly, WDP provides the ability to record performance and available for analysis via Windows Performance Analyzer [2].

4 Method

The subject to be rendered is the Stanford Bunny from Stanford University's 3D Scanning Repository (Figure2). Containing 35947 vertices, the bunny provides an adequate sample to test different point clouds data rendering techniques on the HoloLens 2.

Figure 2. The Standford Bunny [1]

4.1 Importing Data and Project Setup

The original file is in Polygon File Format (.ply) in ASCII. Although both vertices and triangles data are provided, only vertices data is needed. For ease of importing data, the file is imported as a text file, however, to support runtime data loading. Each vertex line is read and added to Unity's Vector3 list, which gets applied to different Unity Rendering Assets. At a high level, each technique is similar(Figure3).

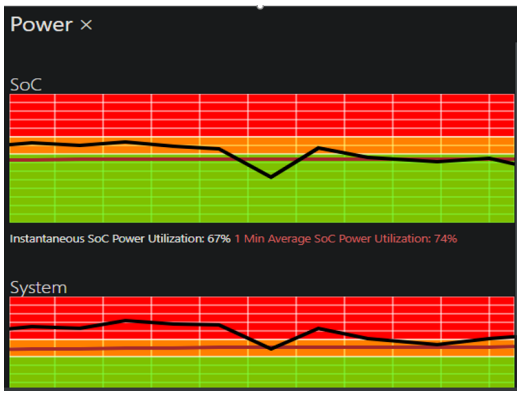
Figure 3. Generalized process from data points to visualization.

To test the two render pipelines, separate Unity projects are set up and applied with Built-in RP and URP, respectively. Each technique is implemented in individual scene of each project to prevent unnecessary overhead and inaccuracies in maintaining components and assets while playing. All techniques are implemented with their default settings, and no measure of optimization was taken.

4.2 Recording

Each scene is built and deployed independently to the HoloLens 2. The Unity Profiler is connected during application execution and the modules are set to GPU, CPU, Memory and Rendering (Figure4a).
While the application plays, Unity Profiler records for 300 frames. The resulting data is then selected for every 10 frames based on the following criteria: CPU and GPU framerates, Memory usage, and Batches Count.

(a) A sample of the Unity Profiler with 4 modules selected. The bottom panel gives details to the selected modules at the current frame.



(b) The SoC Power Utilization Panel.

For power consumption measurement, the application is played for 5 minutes. Every 5 seconds, the Average SoC Power Utilization percentage is recorded. This determines the power utilization trend of each technique(Figure4b).

Finally, a picture of each rendered bunny is taken with the HoloLens 2, for comparison go to Appendix.

5 Results

5.1 Game Objects

Rendering using Game Objects method takes the longest amount of time per frame, 4.7 FPS for CPU, and 25.9 FPS on the GPU for Built-in Pipeline (Table1). However, the same technique performs worse on URP, where the CPU averages at 1.2 FPS and 3.2 FPS for GPU (see Table 1 for average time spent per frame). On the other hand, there is a stark contrast in batches count between the pipelines. Not only is this method's batch counts the highest among the methods, URP also incurred 400 times more batches than its Built-in counterpart (Table1). For each frame, there were the same number of draw calls as there were batches. Rendering over 35k Game Objects simultaneously, URP was unable to either statically nor dynamically batch its draw calls. In other words, it was not able to combine the objects into a bigger mesh for rendering, resulting in individual calls for draw. Draw calls can cause performance overhead on the CPU if not optimized. Although the total used memory of the method overall was significantly higher than the rest, the main cause is the high batches count.

5.2 Mesh Topology

The CPU framerate of Mesh Topology method remains consistently low for both pipelines, indicating minimal computational load on the CPU. For both pipelines, CPU performance is stable at least at 30 FPS (Figure6). In addition, GPU performs well above 60 FPS for both rendering pipelines (Figure7) as well as a stable framerate throughout, providing better user experience. On the other hand, despite the low batches count, it's the same as draw calls count. The draw calls are not being optimized. This could be for a multitude of reasons, shader and material settings, cast and receive shadows settings, as well as the vertices count of the mesh exceeds the limit for Unity dynamic batching. Memory consumption and power utilization do not impose on performance.

5.3 Particle System

Having the best GPU framerate, Particle System can smoothly render over 100 FPS (Figure6), yet suffers CPU bottleneck, especially in the URP, which can render at best 30 FPS. This can potentially be mitigated with optimization settings. However, the batches counts are less than draw calls, for both built-in batching and dynamic batching, indicating some optimization. While dynamic batching builds all compatible batches to a large vertex buffer bound to graphics device, allowing more efficient draw calls, matrix transformation of local object space to world space is done on the CPU [8]. This could explain the overhead observable in (Figure6b).

5.4 Compute Shader

While Compute Shader is thought to have great GPU driven performance, actually did worst than the other methods (except Game Objects rendering) as seen in Figure7. An unexpected occurrence, most of the CPU usage is from scripting, with regular scripts like Mixed Reality Tool Kit scripts taking 3 times as long per frame, contributing to the lower framerate.

5.5 VFX Graph

As Figure6b and Figure7b illustrate, VFX Graph shares similar performance characteristics to Mesh Topology in the same pipeline, except more efficient in power utilization. In addition, batching is not supported for this method either.

6 Discussion

As Figure6, Figure7 and Table1 illustrate, the more suitable render pipeline for the HoloLens 2 is the Built-in Render Pipeline. First, Game Objects rendering method renders significantly better for its efficient draw calls. Secondly, the CPU and GPU framerates are higher and more stable than URP. Thirdly, most rendering techniques are able to render around 30 FPS, which is the standard rendering goal for this device. On the other hand, URP has a slight advantage in lower power utilization percentage, and supports robust visual effect graphs like VFX. As for memory consumption, there is a significant increase with URP, however, the consumption is only a fraction of the overall available memory. In terms of rendering methods, they are all CPU bound. However, Mesh Topology is best for both Built-in Render Pipeline as well as URP. On the other hand, if a developer is looking to develop with just the URP, VFX Graph is slightly more efficient in its CPU performance and power utilization.

7 Future Works

This paper is meant to be ground work to interest developers in optimizing HoloLens 2 performance. There are still many variables to investigate to further understand the HoloLens 2. Future works include, reduce CPU bound with rendering techniques, as the GPU are performing with very high FPS, yet the CPU is the bottleneck. Investigate pipeline settings to optimize rendering performance. Other shader techniques can be explored, such as HLSL or Shader Graph. Create simple unlit materials with the least amound of post-processing, since it is not needed for point clouds rendering. Optimizing current compute shader rendering method, as the current method is highly inefficient for a technique that's theoretically GPU bound. Optimize particle system settings, experiment with GPU instancing. Investigate custom rendering pipeline just for rendering point cloud data in HoloLens 2. There are many potential improvements that could make the URP more efficient. Finally, stress testing with larger point clouds data.

8 Conclusion

In this paper, 5 different point clouds data rendering methods: Game Objects, Mesh Topology, Particle System, Compute Shader, and VFX Graph are implemented on 2 Unity pipelines: Built-in and Universal. They are then executed on the HoloLens 2 and the performance are recorded based on the following criteria: CPU and GPU framerates, batch counts, memory consumption, and power utilization. Overall, the rendering techniques suffer from CPU bounding, resulting in performance bottlenecks by CPU. From the gathered data, the Built-in pipeline is more efficient, especially with Mesh Topology, and URP with either Mesh Topology or VFX Graph. It also goes to show, without any optimization, these methods are still able to reach a good framerate.

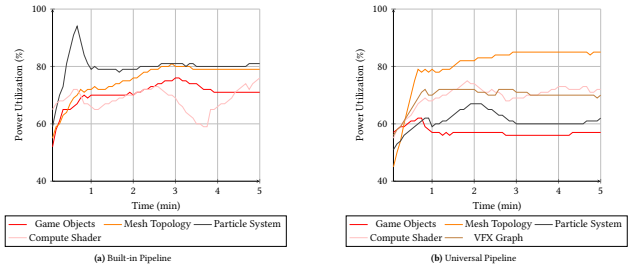


Figure 5. SoC Power Utilization % of the rendering methods between pipelines

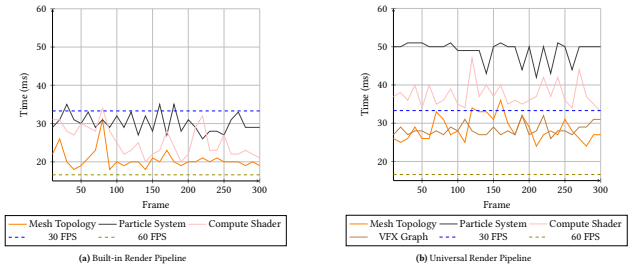


Figure 6. CPU Performance between pipelines. The dashed lines serve as visual guides to show the desired range of framerate. Ideally, for devices such as the HoloLens, the program should aim for a stable 30 FPS. Game Objects has been removed to provide better visual comparison between the rest of the methods.

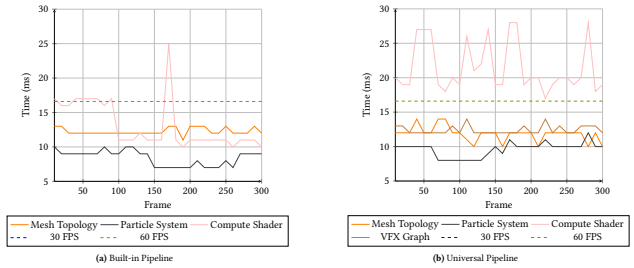


Figure 7. GPU Performance between pipelines. Only a guide line to demonstrate 60 FPS as all methods far exceed the 30FPS mark. However, Game Objects is removed to better exemplify the rest of the techniques.

Table 1. Averaged metrics between Built-in and Universal Render Pipeline across 5 methods. (1): Game Objects, (2): Mesh Topology, (3): Particle System, (4): Compute Shader, (5):VFX Graph.

	Built-in				Universal				
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(5)
CPU Avg (ms)	210	21	30	25	784	29	49	38	28
GPU Avg (ms)	38.5	12	8	13	309	12	10	22	12
Batches Count	50	10	12	9	20021	14	14	14	15
Total Mem (MB)	174	107	162	119	232	150	143	136	161
Power Util (%)	70.5	75.4	79.9	68.4	57.1	80.3	61.1	69.8	69.7

References

[1] 1994. *The Stanford 3D Scanning Repository* (1994). http://graphics.stanford.edu/data/3Dscanrep/

[2] Hamalawi, hferrone, reburkea, cre8ivepark, joyjaz, and DCtheGeek. 2020. Using the Windows Device Portal - Mixed Reality. *Mixed Reality | Microsoft Docs* (Aug 2020). https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/using-the-windows-device-portal

[3] Manorama Jha. 2020. How to MR - Chapter 1: Alignment in Mixed Reality. *Manorama Jha Manorama Jha* (Dec 2020). https://manoramajha.medium.com/alignment-in-mr-475444cba3ec

[4] Markus Schütz, Bernhard Kerbl, and Michael Wimmer. 2021. Rendering Point Clouds with Compute Shaders and Vertex Order Optimization. https://www.cg.tuwien.ac.at/research/publications/2021/SCHUETZ-2021-PCC/

[5] Scooley, Seth Paniagua, v jmathew, joyjaz, evmill, and v jodben. 2020. HoloLens 2 hardware. https://docs.microsoft.com/en-us/hololens/hololens2-hardware

[6] scooley, v jmathew, joyjaz, evmill, v jodben, and SethPaniagua. 2020. HoloLens 2 hardware. *HoloLens 2 hardware | Microsoft Docs* (Oct 2020). https://docs.microsoft.com/en-us/hololens/hololens2-hardware

[7] Shahriar Shahrabi. 2020. Point cloud rendering. *Medium* (May 2020). https://medium.com/realities-io/point-cloud-rendering-7bd83c6220c8

[8] Unity Technologies. 2021. Draw call batching. https://docs.unity3d.com/Manual/DrawCallBatching.html?_ga=2.130008460.1468640980.1628158259-2435465 14.1621552421&_gl=1*yo43cl*_ga*MjQzNTQ2NTE0LjE2MjE1NTI0MjE.*_ga_1S78EFL1W5*MTYyODIwMzI0My4xNTEuMS4xNjI4MjA0NzQ0LjYw

[9] Unity Technologies. 2021. Feature comparison table: Universal RP: 10.5.1. *Universal RP | 10.5.1* (Jul 2021). https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.5/manual/universalrp-builtin-feature-comparison.html

[10] Unity Technologies. 2021. High Definition Render Pipeline/Built-in Render Pipeline comparison: High Definition RP: 12.0.0. *High Definition RP | 12.0.0* (Jul 2021). https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@12.0/manual/Feature-Comparison.html#vr

[11] Unity Technologies. 2021. Particle systems. *Unity* (Jul 2021). https://docs.unity3d.com/Manual/ParticleSystems.html

[12] Unity Technologies. 2021. Profiler overview. *Unity* (Jul 2021). https://docs.unity3d.com/Manual/Profiler.html

[13] Unity Technologies. 2021. Render pipelines introduction. *Unity Documentation* (Jul 2021). https://docs.unity3d.com/Manual/render-pipelines-overview.html

[14] Unity Technologies. 2021. Scriptable Render Pipeline introduction. *Unity Documentation* (Jul 2021). https://docs.unity3d.com/Manual/scriptable-render-pipeline-introduction.html

[15] Dorin Ungureanu, Federica Bogo, Silvano Galliani, Pooja Sama, Xin Duan, Casey Meekhof, Jan Stühmer, Thomas J. Cashman, Bugra Tekin, Johannes L. Schönberger, Pawel Olszta, and Marc Pollefeys. 2020. HoloLens 2 Research Mode as a Tool for Computer Vision Research. *CoRR* abs/2008.11239 (2020). arXiv:2008.11239 https://arxiv.org/abs/2008.11239

A Built-in Render Pipeline Rendered Rabbits

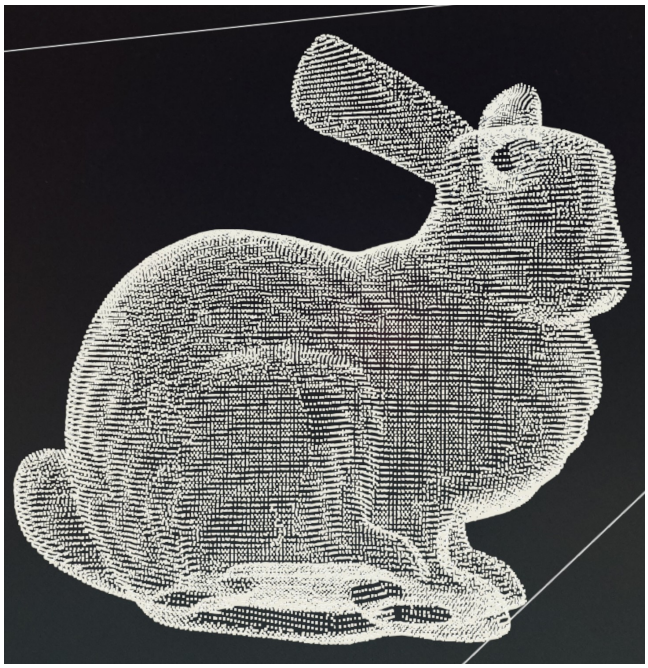


Figure 8. Game Objects method.

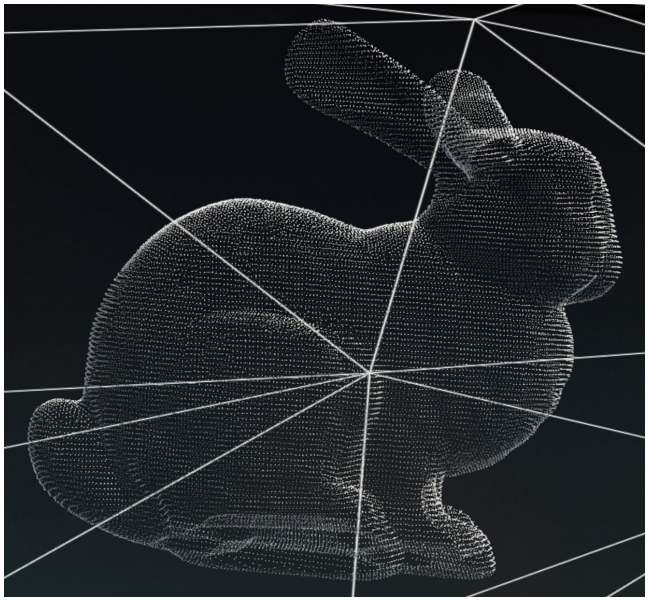


Figure 9. Mesh Topology method.

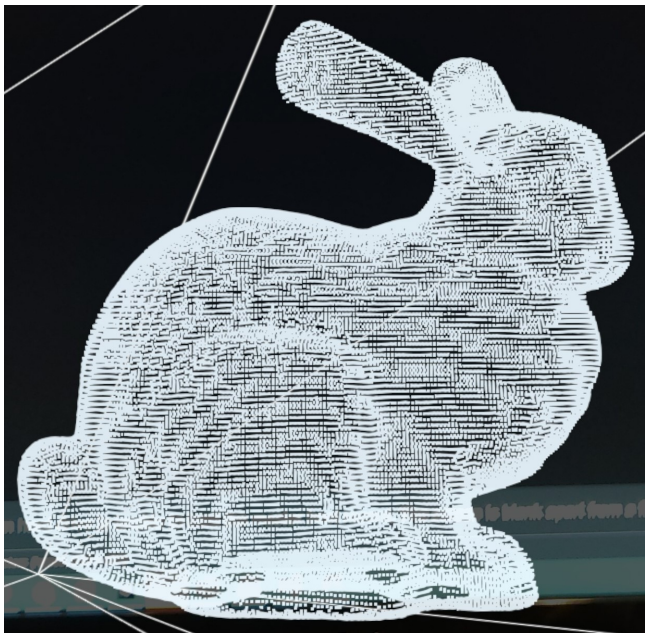


Figure 10. Particle System method.

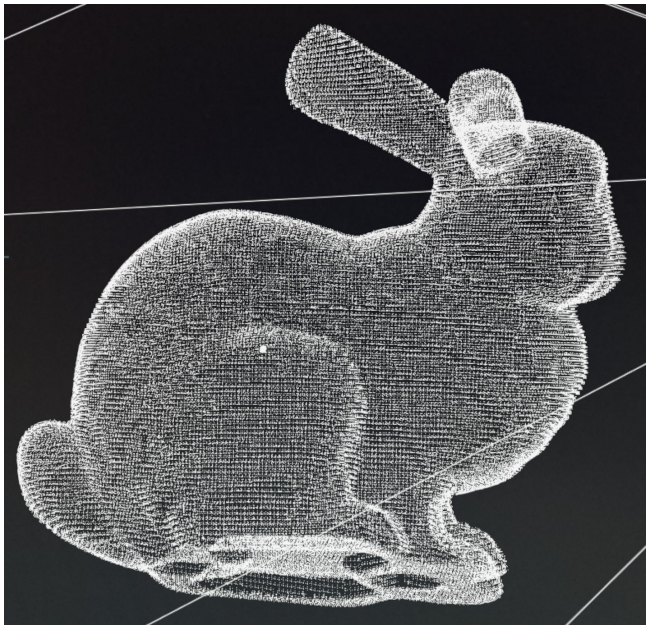


Figure 11. Compute Shader method.

B Universal Render Pipeline Rendered Rabbits

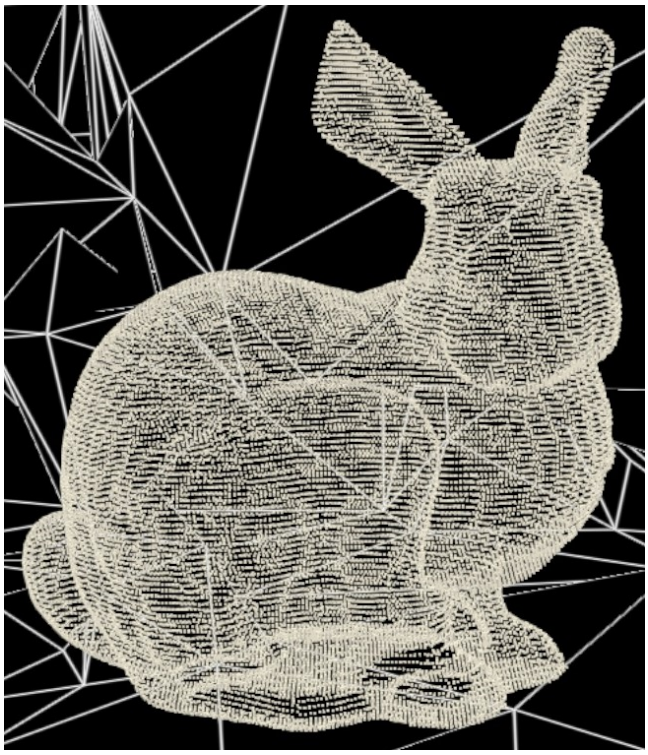


Figure 12. Game Objects method

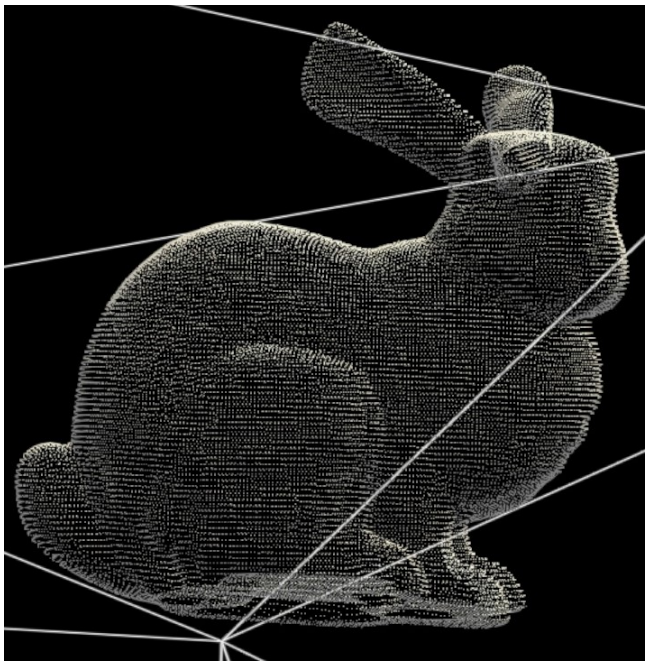


Figure 13. Mesh Topology method.

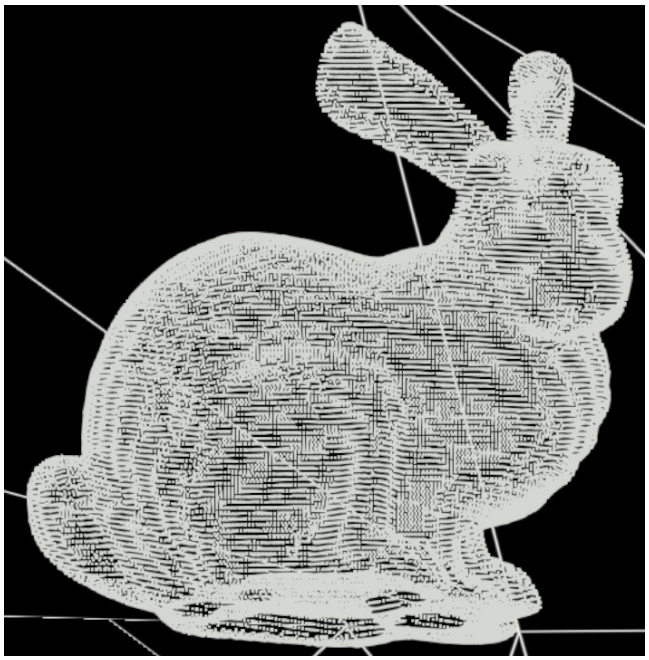


Figure 14. Particle System method.

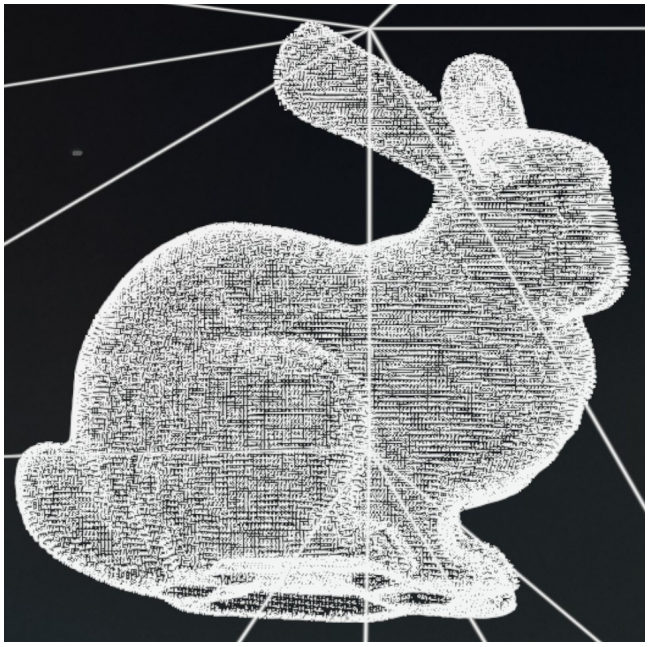


Figure 15. Compute Shader method.

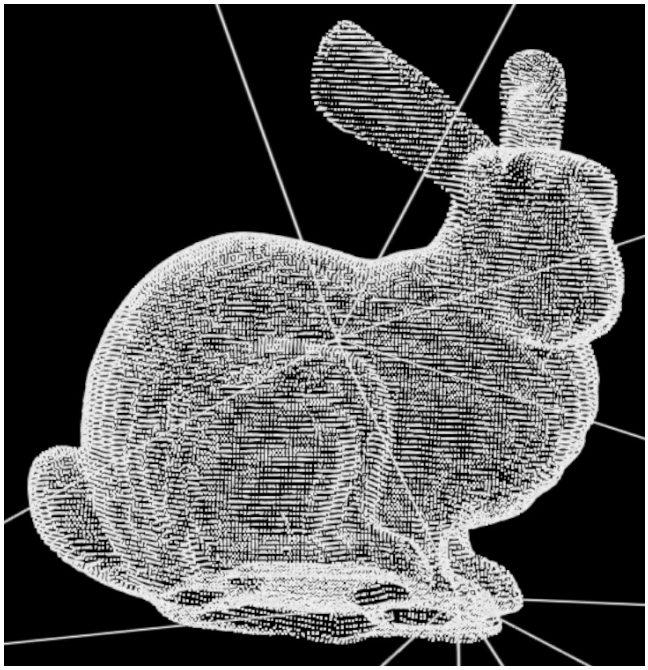


Figure 16. VFX Graph method.