5-2024

LODs and Nanite within Unreal Engine 5: The Future of 3D Asset Creation for Game Engines

Stephen R. Overton
East Tennessee State University

LODs and Nanite within Unreal Engine 5:

The Future of 3D Asset Creation for Game Engines

A Culminating Experience

presented to

the faculty of the Department of Digital Media

East Tennessee State University

In partial fulfillment

of the requirements for the degree

Master of Fine Arts in Digital Media

---

by

Stephen R. Overton

May 2024

Jacy Richardson, Chair _____

Todd Emma _____

Cheryl Cornett _____

Keywords: unreal engine 5, nanite, LODs, modular assets kits

ABSTRACT

LODs and Nanite within Unreal Engine 5:

The Future of 3D Asset Creation for Game Engines

by

Stephen R. Overton

In the video game industry, developers utilize game engines to bring their creations from concept to reality. However, the most widely used engine in the industry currently is Unreal Engine 5 or UE5, which has challenged established practices used by game developers since the early creation of 3D interactive media. One of these challenges is eliminating LODs or Level of Detail-based asset integration with the introduction of Nanite, an automatic LOD creation tool introduced in Unreal Engine 5. With this development, it is still being determined whether Nanite should immediately replace LODs due to its ability to cut out work required for LOD-based integration. This uncertainty has led to the purpose of this study, which is to research and understand the background and utilization of LODs and Nanite in 3D game asset creation while showcasing how both processes intertwine inside UE5.

The following research questions will guide this study in answering questions and setting up foundation knowledge into LODs and Nanite to understand the importance of each optimization technique and why their usage matters to the future of 3D asset creation. First, what are LODs, and why are they used in 3D asset creation within the video game industry? Second, what is Nanite, and how does this process differentiate itself from the methods utilized in LOD creation? Third, what are the benefits and consequences of using either LODs or Nanite in 3D asset creation? Lastly, can both processes be utilized in tandem inside Unreal Engine 5.2.1 to allow developers to use the best abilities of both methods in their 3D asset creation pipelines?

DEDICATION

For Nichole and Stephen, a.k.a. Mom and Dad. You both have pushed, taught, and given me experiences throughout my life that have shaped me today as an artist and person, and I will always be thankful for these times now and forever.

For Elisabeth, you have been a constant support pillar throughout my life. I am deeply thankful for our bond and for having a sister who has always been there for me just as I have been there for you.

For Weston Hopper, you were one of the best parts of coming straight into the M.F.A. program from undergrad. I will never forget our time as desk mates when we talked and discussed films and games as we worked on our artwork while having movies playing in the background.

For my fellow grad students, our time together has been a treasure. We have worked, laughed, and enjoyed each other's company, formed memories, inside jokes, and a deep connection as we braved through our M.F.A journeys. I eagerly anticipate the heights and paths we will take in the future, knowing that we all have the potential to achieve anything we set our creative minds towards doing.

Lastly, this paper and project are in remembrance of my late grandmother, grandfather, and uncle Edith "Blondie" Overton, James R. Overton, and James "Derek" Overton, who provided me with a lifetime of memories, helped start me down the path of pursuing professional video game development and gave me the drive to seek out and complete the most challenging obstacles I have tackled in my life, no matter the difficulty.

ACKNOWLEDGMENTS

I want to express my gratitude to the committee chair, Jacy Richardson, for always supporting me in my growth as a mentor in 3D art, pushing me to improve myself both academically and professionally, and telling me when I asked for feedback, “Hmmm I think it needs more trash.” I also want to extend this gratitude to my fellow committee members, Todd Emma and Cheryl Cornett, who have extended their expertise to help aid me in my research and professional practices for this project and my professional development as a 3D artist. I want to extend gratitude to the rest of the ETSU Digital Media faculty as I have learned so much from you all, which has shaped me into the artist I am today. Lastly, I want to thank Dr. Joe D. Moore and Dr. Dennis Depew, who allowed me to pursue my studies further in graduate school and reach this point in my academic journey.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1. Introduction

Within the game industry, the production requirements of high-quality and optimized 3D assets are of top priority for developers in developing games ranging from indie to triple-A (AAA) due to developers being tasked with creating 3D assets that retain high amounts of graphical fidelity while still being optimized for use within game productions. In addition, developers are usually limited in how they can optimize their games because they must tailor the gameplay experience for their lowest-targeted platform while retaining more significant fidelity details for higher-end platforms, such as personal computers (PCs). With this need for high-quality assets and limitation based upon targeted platforms, developers must utilize optimization methods to efficiently control the number of details present in objects to optimize the gameplay experience in visual and technical performance to improve the overall player experience. The current design technique for this task is Level of Detail (LODs), which requires multiple instances of static meshes and environments that are swapped out to lower-quality versions of themselves depending on how close the player's renderable viewing aera is to said object. Recently, in Epic Games's release of Unreal Engine 5, a new technique of asset optimization and rendering has emerged called Nanite, which utilizes the base principles of LODs but differs by only using the initial object to perform procedural decreasing of details and rendering the areas of the asset the player can visibly see during gameplay.

With the introduction and implementation of Nanite in UE5, the future use of LOD assets has become unclear in the greater gaming industry. At the same time, information about Nanite is limited to Unreal Engine 5's documentation and showcase videos published directly by Epic Games. This uncertainty in both Nanite and the future of LODs in the game development industry has led to the purpose of this study which is to research and understand the background

and utilization of LODs and Nanite in 3D game asset creation while showcasing how both processes intertwine with each other, regarding 3D assets, inside of Unreal Engine 5.2.1. In addition, the methodology that will be used to highlight this paper research findings will utilize my pre-existing professional skillset in 3D game asset creation by making a traditional environment asset kit, with Nanite in mind, and displaying this kit by using it to create a small showcase environment inside of Unreal Engine 5. In doing this, the goal will be to display what pros and cons come from making assets for Nanite in mind, what limitations there are when developing for Nanite vs. LODs, and what takes aways there are when creating assets for use with Nanite inside of Unreal Engine 5.

Chapter 2. Literature Review

Developers must solve the problem of maintaining a higher amount of graphical fidelity while still optimizing 3D assets so that user experience during gameplay is not negatively impacted, no matter what the targeted platform is. From this need to balance detail and performance, developers utilize the method of LODs as the primary way to balance these two aspects of 3D asset creation for video games. LODs are described as instances or copies of a static mesh that are manually or procedurally lowered in terms of polygon count and texture detail and then swapped in and out during gameplay depending on a set percent distance of the virtual camera or the player's view to the LOD object or environment (Methods and Systems, 2022). In using LODs for asset optimization, game developers can utilize the initial high-detail assets and textures in combination with lower visual copies to create game environments that hit the optimization requirements of less powerful platforms while still delivering higher visual quality and performance to more powerful platforms.

Although using LODs helps developers optimize 3D assets and environments, their inclusion can improve a game's performance if the developers appropriately integrate them properly. For instance, when creating LODs, developers must decide how much fidelity show will be present on each LOD and at what percentage or distance each LOD should be loaded in and swapped with the current mesh during gameplay. If developers do not correctly set up the percent values in which higher or lower-quality static meshes should be loaded, they could risk pixel popping during gameplay. Pixel popping happens when the switch between each LOD mesh is apparent and happens in almost snapping movement instead of a smooth transition between each LOD model (Methods and Systems, 2022.) In addition to potential pixel popping, when creating LODs meshes, developers must account for the initial model's fidelity and

positioning inside of the game's environment to ensure that the copies retain the defining features of the initial mesh while also being able to properly represent the mesh from any position it could be viewed during gameplay (McRoberts & Hardy, 2007). In the case that the model position is not considered when setting up LOD thresholds, developers will run the risk of incorrect LOD levels being loaded in at the untended distance due to improper swapping percentages that do not account for all areas surrounding the mesh, such as its height or positioning in the environment, harming the overall player experience of the game. LODs are powerful tools when used correctly, allowing developers to deliver games with high performance and visuals to various platforms. However, a new model optimization method recently introduced to Epic Games's Unreal Engine 5 has changed how 3D assets are optimized as it is an evolution of the concept of LODs.

Epic Games best describes Nanite as a virtualized geometry system that renders static meshes based upon what is seen in the camera view and does not render unseeable details while creating automatic LODs transition within only the initial static mesh (Nanite Virtualized Geometry, 2022). When utilizing Nanite inside of Unreal Engine 5, its visual core is reliant on triangles, which is the base representation of any 3D model, because the goal with Nanite is not to disrupt established industry art production pipelines or software toolsets but rather to improve current methods in geometry rendering and processing for in-game assets and using triangles ensures that developers can utilize all of the features with Nanite without having to change their existing production pipelines. (SIGGRAPH, 2021). In utilizing triangle clusters, Nanite differs from traditional LODs in two ways: by only rendering clusters that are renderable in the current camera view and by actively selecting the maximum LOD amount per cluster, meaning that it will only pick specific clusters to give a greater detail LOD to depending on the player's view in

the level which saves on both storage memory and rendering power during gameplay (Nanite in UE5, 2021).

When discussing Nanite, the process adopts traits synonymous with LODs but has distinctive properties that make it stand out as a geometry rendering and processing system. For example, Nanite can process and render static meshes comprised of millions upon millions of polygons without dipping in gameplay or rendering performance, which is impossible when utilizing LODs due to the constant loading and unloading of static meshes during gameplay (Nanite in UE5, 2021). As an example of this, authors Zhong et al. (2021) proposed a new method of implementing Nanite into their pipeline for processing high poly 3D mesh reconstructions of images, commonly known as photogrammetry, by utilizing Nanite on a reconstructed mesh directly from Reality Capture (Zhong et al., 2021). With the implementation of Nanite, Zhong et al. (2021) were able to eliminate a significant portion of their established photogrammetry production pipeline that included modeling practices such as retopology and texture baking where an artist would have to take the high-quality outputted mesh from Reality Capture and manually lower its polycount for use in LOD creation, while also handling baking the initial texture maps back onto the newly lower polygon models (Zhong et al., 2021). In using Nanite in their pipeline, Zhong et al. (2021) were able to cut out a sizeable portion of work by directly importing their high poly reconstructed meshes from Reality Capture into Unreal Engine 5 due to Nanite's ability to create small clusters of LODs automatically without the immediate need for asset optimization that is commonly seen with traditional LOD creation. With the introduction of Nanite, a new evolution in asset optimization, processing, and rendering has been established that has the groundwork for future development in perfecting how assets are optimized for use within Unreal Engine 5. However, both Nanite and LODs have their respective

benefits and consequences when used during 3D asset creation, and thus, pushes developers to properly understand and recognize when and where to utilize either Nanite or LODs.

Developers must consider certain benefits and limitations when implementing either process in their development pipeline. For instance, when creating custom models or actors for an environment, it is currently beneficial for developers to utilize LODs over Nanite. The reasoning for this choice is that LODs allow developers more freedom in the types of actor actions available for them to use, where Nanite does not support specific actor actions such as mesh deformation or non-opaque materials, thus limiting the choices developers have when utilizing this toolset over LODs. As an example, during *The Matrix Awakens: Generating a World* showcase, the development team created custom deck pieces that would fit the organic curvature of the roadways, making instancing almost impossible due to each piece being custom-made for the given curvature and the need to be potentially destroyed (Generating a World, 2022). In using custom static meshes, LODs are better suited as they retain the ability to be deformed or destroyed while retaining the optimization benefits by swapping out with higher or lower variations, which Nanite currently does not support in Unreal Engine 5.2.1.

In contrast, if the created static mesh is designed solely for decoration purposes, is repetitive in the environment, or does not need the ability to be destroyed or deformed, Nanite would be the better choice since it can handle instances of high polygon statice meshes and actors without any negative consequences to performance both in the editor and during gameplay. For example, during the *Building Open Words in Unreal Engine 5* showcase, Unreal Engine Instructor Sam Deiter discusses a feature called packed level actors, which takes current objects within a level and replaces them with instanced static mesh copies and then groups them into a single actor for more manageable level editing and smoother processing during gameplay

(Building Open Worlds, 2022). In addition, Deiter demonstrates this tool by combining 37 ground tile static mesh actors, with Nanite enabled on each mesh, into a single pack level actor to efficiently group them, thus saving on resource space, and showcase how developers can use packed actors with Nanite to create detail environment with little time and effort (Building Open Worlds, 2022).

In using repeatable static meshes, Nanite would be a better fit for this task than LODs because multiple copies of high poly static meshes are combined into a single packed actor, which is then further copied on its own and placed around an environment. In addition, a developer would save on overall environment resources and rendering power since Nanite can handle duplication of high poly static meshes and could help control the overall rendering costs of the ground tiles due to it determining when finer details are needed and not needed during gameplay. With these few examples, both LODs and Nanite have strengths and weaknesses in using them for asset creation and development inside and outside of game engines. However, the idea of LODs and Nanite being used together is something developers must consider due to each process benefiting from each other's strengths while covering each other's shortcomings. In discussing the consequences and benefits of using either LODs or Nanite during asset and environment creation inside Unreal Engine 5, it is shown that both LODs and Nanite have their respective uses inside Unreal Engine 5. However, the question remains if it is possible to utilize both processes simultaneously to allow developers to benefit from both toolset's strengths while using them to cover each other's shortcomings.

During *The Matrix Awakens: Creating a World* tech talk, hosted by Technical Art Director Jerome Platteaux from Epic Games, the concept of simultaneously utilizing both LODs and Nanite is explored when Platteaux discusses how the development team behind the "The

Matrix Awakens: An Unreal Engine 5 Experience" set up their city environment inside of Unreal Engine 5. For example, Platteaux's team started their open world setup by utilizing Hierarchical Level of Detail (HLOD) grids to control how actors are loaded and unloaded in memory during gameplay; for this instance, the team created three grids named main grid, HLOD0, and HLOD1 based upon the technical and visual complexity of each grid aera (Creating a World, 2022). HLOD toolsets are systems that use custom HLOD layers, broken up into cells at runtime, to organize static mesh actors into a single generated mesh and material, thus increasing performance in large open-world environments (World Partition, 2022).

The main grid level, as described by Platteaux, is a 128m loading range comprised of nine cells surrounding the player that loads all actor types placed within the level, such as static mesh buildings, props, and collision boxes, while also noting that anything changes with the main grid level would cause changes to HLOD 0 and 1 which would lead to recalculations for both HLOD levels (Creating a World, 2022). For HLOD0, it starts after the 128m range of the main grid level and stretches to a 768m loading range, with each cell within HLOD1 being comprised of four individual cells from the main grid aera while having four unique actors that combine objects and turns them into Instant Statice Meshes (ISMs), which are instances or copies of existence meshes present in the level environment, to help save on resources used during run time (Creating a World, 2022).

In addition, HLOD0 uses data layers to control which elements from the main grid aera are present with each of its cells to lower the actor count while improving the overall loading and rendering of each of its cells without sacrificing detail for the player experience (Creating a World, 2022). Lastly, with HLOD1, its cell size is everything in the level loaded past the 768m boundary found in HLOD0 and is always present in the game's memory by combining all present

actors into a singular Nanite mesh, which creates a LOD that will not take up additional memory and rendering due to using a singular Nanite mesh to represent the furthest outward point from the player's position in the level (Creating a World, 2022). With Platteaux's team's example of utilizing HLODs and Nanite within the same environment, developers can implement both process without worrying about the adverse side effects associated with each process due to them complementing each other's strengths both in the level editor and during gameplay.

With the research done into LODs and Nanite, both processes have substantially impacted 3D asset development for use within game engines. However, both LODs and Nanite have negative consequences, such as potential asset pop-in and out seen with LODs or Nanite being only able to support opaque actors and not handling deforming or destructible static meshes. With these downsides, developers must understand when and where Nanite or LODs should be implemented to benefit their game environment's performance and fidelity. However, if this line of caution is exercised, 3D asset and world developers now have access to two extremely powerful development toolsets, and if used in conjunction, the types of environments utilizing them will be able to become almost limitless in terms of overall detail and performance capability.

Chapter 3. Research Methodology

In researching the benefits of LODs and Nanite, the main point I found was a lack of direct resources comparing the two processes together by utilizing the same 3D assets to showcase what effects and practices need to be implemented when creating, developing, and implementing finalized assets into an Unreal Engine 5 environment. With a lack of direct comparison showcasing LODs and Nanite in professional studies, I started researching the best professional methods to showcase the development practices required to showcase LODs and Nanite being used together inside Unreal Engine 5. In my research, I found that the professional practice of creating a modular asset kit would serve my needs to showcase the difference between the use of LODs and Nanite, which would lead back to the goal of this project of showing the benefits and consequences of using the process inside of Unreal Engine 5. From here, I shifted my focus towards deciding what theme would be utilized to create my modular asset kit for testing LODs and Nanite in Unreal Engine 5.

When deciding upon the theme and workflow for this project's modular asset kit, I researched what types of modular kits are used professionally in the industry to establish a desired workflow for how I should develop the modular asset kit for researching and demonstrating both LODs and Nanite. During this process, I discovered a professional modular asset kits company called KitBash 3D, which specializes in creating modular asset kits with varying themes and detail levels for use in film and game environments. After reviewing their professional kits, I decided to use their approach to asset kit development to serve as the foundational guide for what I wanted to emulate in my workflow when creating the project's experimental modular asset kit. Following the decision to use KitBash 3D as a foundational guide for this project's kit, the next step was to decide on what theme would be used during

development since asset kits utilize a uniform theme, showcased in Figure 1, to ensure the asset kit is consistent in its overall art style, level of details present in its assets, and overall usability and versatility of the finalized assets within the intended game engine.

Figure 1: Modular Asset Kit breakdown from KitBash 3D's Dark Fantasy Kit (KitBash 3D)

I started researching potential themes by developing mood boards of possible environments that both fit the asset kit conditions and showcase my understanding of professional practices in modeling, texturing, and asset implementation inside of Unreal Engine 5. The chosen environment for this showcase was Amsterdam, the capital of the Netherlands, showcase in Figure 2. In using Amsterdam as the asset kit's theme would provide an overall art style in the asset kit theme would provide an overall art style in the kit and the environment having a unifying level of detail in the architectural structures and environment design that is unique to Amsterdam, and the cities structures lend themselves to a high level of usability as they can be broken into modular parts and sections as seen in Figure 1. When deciding upon Amsterdam for the asset pack's theme, the main conditions that were considered were, "Could the environment handle the three defining features of an asset kit?" and "Could the environment

provide a unifying level of detail in asset creation as it relates to the needs for showcasing LODs and Nanite in Unreal Engine 5?”

Figure 2: European City Moodboard for Modular Asset Kit’s Theme

After finalizing the theme for the project's asset kit, the last stage of the pre-development work was to break down the proposed asset kit into what assets needed to be made, gather references for each listed asset type, and create what would be used within the final environment inside Unreal Engine 5. The first step of this process was to break the bulk of the kit into different sections, such as modular buildings, decorative props, and environment assets, such as roads, bridges, and foliage. After breaking down each asset type into its respective group, the next step was to decide what would be made and gather references to plan out each asset type. For example, when deciding upon the buildings, I decided to limit the kit's building amount to between 6 and 8 base models to allow for various building types while ensuring that I could

produce a baseline level of quality that would be shared across all the created buildings. From there, I started gathering reference images for each asset to be used in the creation phase, including images showcasing close-up details for each selected building asset, as seen in Figure 3. After collecting the references for each building type, the last step was to decide how I would implement each asset type into the final showcase, which, for the buildings, would be by creating a rough environment outline, seen in Figure 4, which provided with an initial sense of scale and what should be designed to showcase the finalized asset kit. Once I finalized the usage of each asset type in the final environment, the focus switched away from pre-development to full-scale development, which covers the creation of the kit's 3D assets, the challenges and limitations learned during the creation process, and what was needed to get the 3D assets ready for use with Nanite inside of Unreal Engine 5.

Figure 3: Reference Image of Amderstdam Buildings Utilized in Development

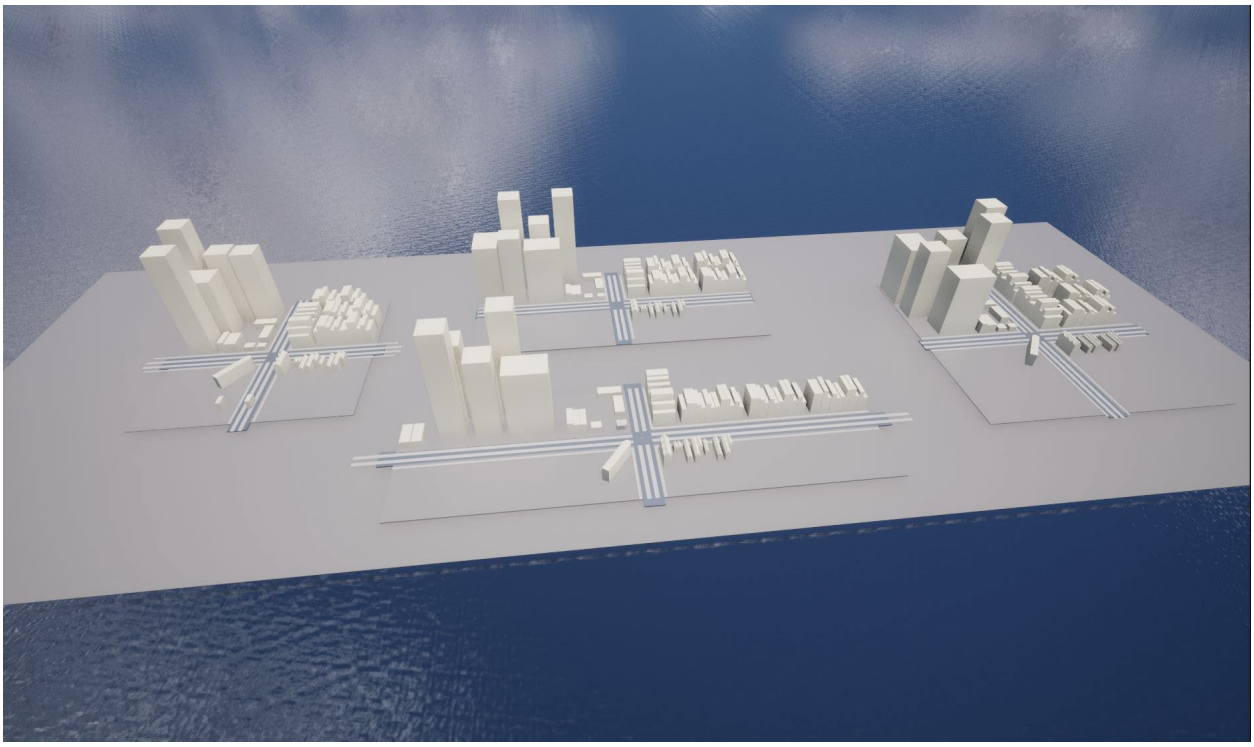

Figure 4: Developmental Building Testing Grayout Image

After I finished the initial research for the asset kit, the development cycle switched to creating the kit's 3D assets, which included making 3D models and textures of the previously listed buildings, decorative props, and environment assets. The first step was to take the gathered reference images and break down each unique section of each model to provide a guide on each of the unique parts that would make up a model, as seen in Figure 5. This step helps to speed up the development cycle, ensure consistency across the entire asset, and simplify customization, which is crucial when dealing with modular assets such as the kit's buildings. After completing the highlight breakdown, I started modeling each unique part from the highlighted images and utilized each part to build the finalized building asset, as seen in Figure 6. For the more commonly used building assets, I would utilize a process called kitbashing, which is a process that originates in the 1970s, popularized by the film *Star Wars IV: A New Hope,* where parts from physical model kits, such as trains, cars, etc. would be mixed or “bashed” to create unique

variations that differed from the initial models. With the concept of kitbashing, I utilized the initially completed building models to make both non-kitbashed and kitbashed variants, allowing for unique variations without the need to model completely new buildings, thus increasing the overall number of buildings for use in the environment.

Once the 3D models were completed, I started finalizing the topology levels for the models to be used with LODs and Nanite. To start this process, I would take the base unsmooth model, LOD 0, seen in Figure 7, and make two variations, LOD 1 and 2, by increasing the levels of smoothness, which would subdivide the topology level of the model to hold its high detail form when exported for texturing and implementation inside of Unreal Engine 5. In implementing this process, I was able to not only make traditional LOD states for use in Unreal Engine 5, but I was able to make variations that were ready for use with Nanite since LOD 2 had the necessary topology amount needed for the automatic LOD creation done with Nanite. With this step completed, the 3D models were ready for the second part of my creation pipeline, where I would texture each asset before integrating them for testing in Unreal Engine 5.

Figure 5: Aaron Church Broken Down Reference Image (PICFAIR, Artur Bogacki)

Figure 6: In-Development Shaded Render Building Breakdown inside Autodesk Maya

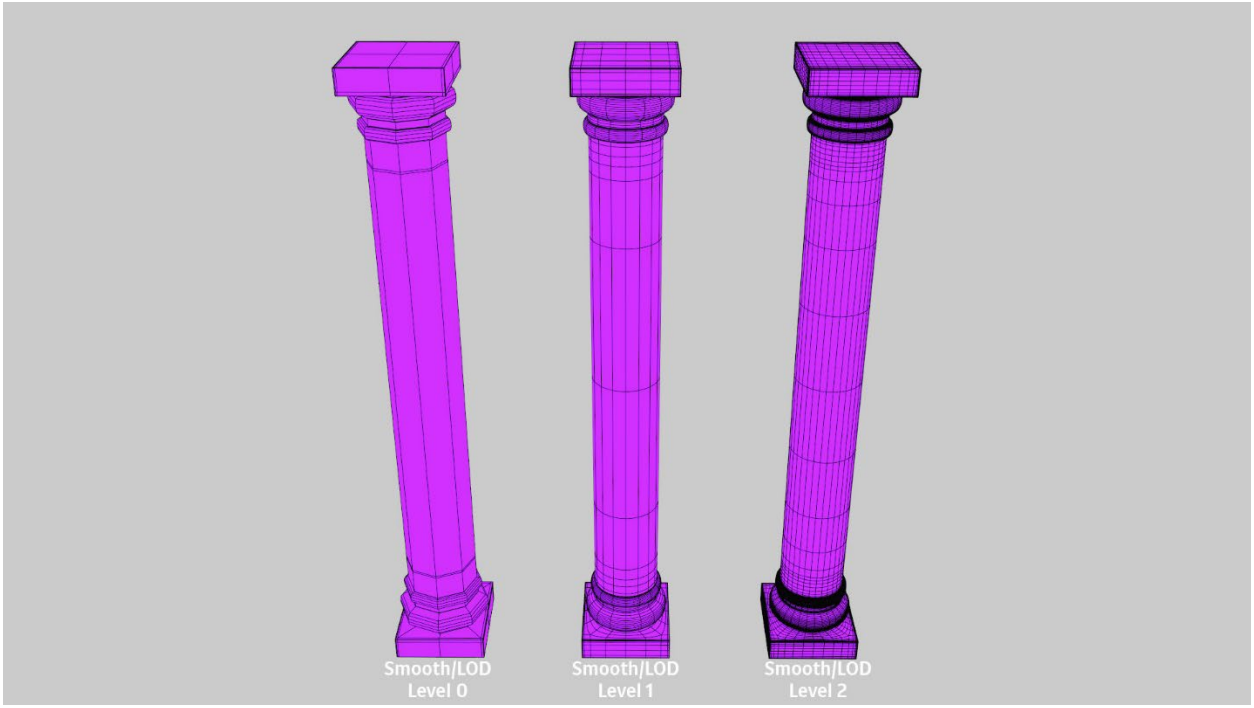

Figure 7: Model Topology Breakdown Smoothness/LODs Levels

Once I completed the LOD versions for the building assets, the next step was to transition from 3D model creation to making textures for each asset type. When I started the texturing process, I kept two main points in mind when designing the textures for each 3D asset within the modular kit. The first was to create materials that could be reusable, so when creating each asset texture, I set up each asset to allow for quick and unique texture variations to be made based upon aspects such as base colors, alpha map decals, surface roughness, etc. The second point I focused on was creating lower-resolution texture variants of each texture type corresponding to the LOD levels, shown in Figure 7, to show the effects of an asset utilizing LODs for display in-engine. To do this, I used a base 2K resolution texture map for LOD level two and half the resolution for each lower LOD state, where I would get a 1K and 512x map for LOD level one and zero, respectively. After completing the textures, following the points above, I transitioned into the last phase of development, where all previously created aspects of the modular asset kit would be implemented into Unreal Engine 5 for testing to see the difference between LODs and Nanite versions of the same 3D models.

After the completion of the asset kit, the work process transitioned from researching professional showcase practices for LODs and Nanite to researching a general theme for the modular asset kit in Amderstdam, to the full development of the asset kit's models and textures for testing, demonstrating, and comparing LODs and Nanite together inside of Unreal Engine 5. Following the creation of the assets, the next aspect of this project is to use this kit to not only showcase the use of LODs and Nanite but also to research and understand the unique challenges for developing assets for these two processes, what needs to be kept in mind for future developers when working with these two processes in-engine, and what can be taken away for

those who wish to future explore and understand the benefits of using both LODs and Nanite in conjunction with one another.

Chapter 4. Data Analysis

With the completion of the modular asset kit, the project switched from the development of the kit's assets to the testing and showcasing phases, where the assets would be experimented with to understand the similarities and differences with implementing assets for use with LODs or Nanite into a game environment along with seeing what unique situations need to be accounted for when using these processes separately or together in Unreal Engine 5. During the testing phase tests, the main points that will be evaluated will be the implementation of each asset type, what each type requires in terms of resources from both files and engine performance and the limitations LODs and Nanite have when compared to each other inside of Unreal Engine 5. Following the testing phase, the kit will be used to create a small environment showcase using the LOD level two assets, with Nanite enabled, to see how groups of higher polygon assets perform during runtime when utilizing Nanite.

Throughout the testing phase, the topics covered will help answer what parts are required for each process's implementation into an Unreal Engine 5 environment, what resources are needed to use LODs and Nanite properly, and what current limitations exist when using LODs and Nanite. For the first topic, as seen in Figures 8 and 9, the required amount of assets for LODs can become relatively high when considering the number of models, textures, and materials needed to create each unique LOD level necessary for proper optimization within a given environment. However, when using Nanite, the required amount of assets decreases to the base textures and models needed for the highest LOD level because Nanite will automatically create the lower LOD levels necessary for proper optimization during gameplay. For example, in Figures 10 and 11, the effects of using LODs can be seen where each LOD level would require separate variations of the assets seen in Figures 8 and 9 and require developers to develop,

implement, and optimize three or more variants for each given asset within the environment. In contrast, if the developer utilizes LOD Level 2 as a Nanite mesh, the number of required models, textures, and materials would decrease to the amount needed for one instance of the finalized asset rather than multiple variations seen with LODs.

With this example, when using LODs, developers would be required to create more assets, allocating more developmental resources to handle each LOD asset variation regarding its creation, in-game implantation, and internal and visual performance management during gameplay. If LODs are utilized, developers are forced to spend time creating unique variations of every asset, texture, and material multiple time multiple times per asset used in-game, which takes away from developmental time for polishing each asset, internal resources within the given game engine, and external resources such as budgets, time constraints, and development timelines for the given project. In contrast, if Nanite is implemented, developers can utilize their established workflow to make the highest LOD level for their given project, such as LOD level 2 in Figures 10 and 11, which would cut down on the points made above regarding LOD differing costs while enabling developers to focus more resources and development time to the creation of higher detailed models, textures, and materials, which allows for an overall higher fidelity to be reached without adding more resources that come with traditional development with LODs.

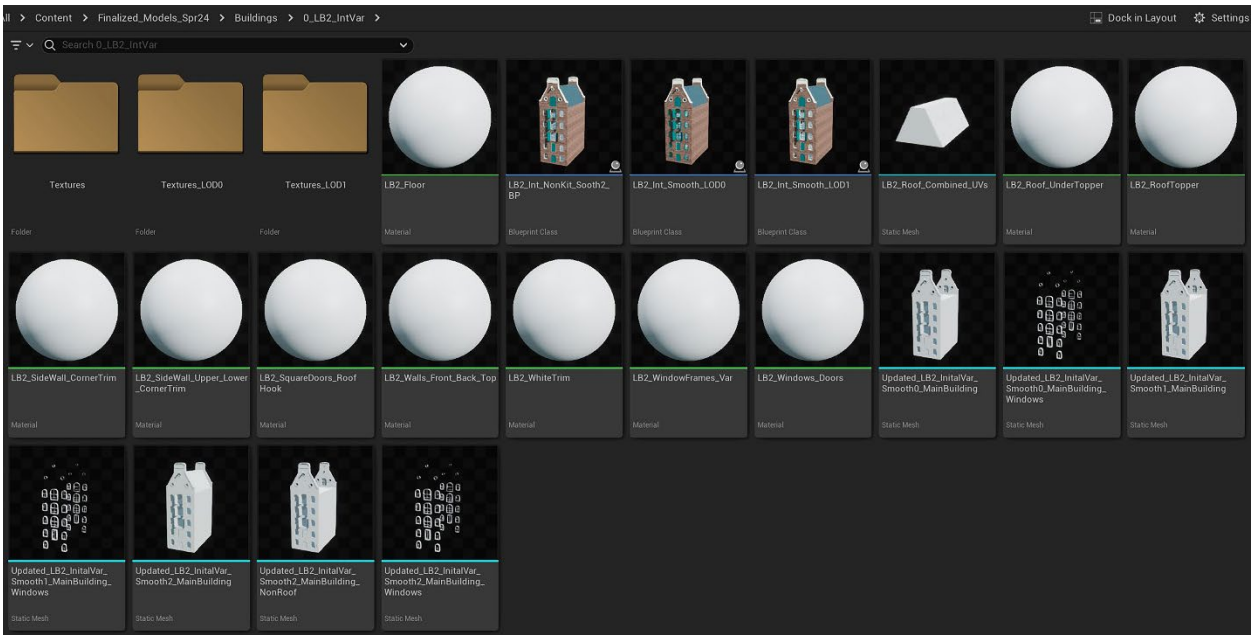

Figure 8: Asset Amount Showcase for LODs and Nanite: 3D Model Parts

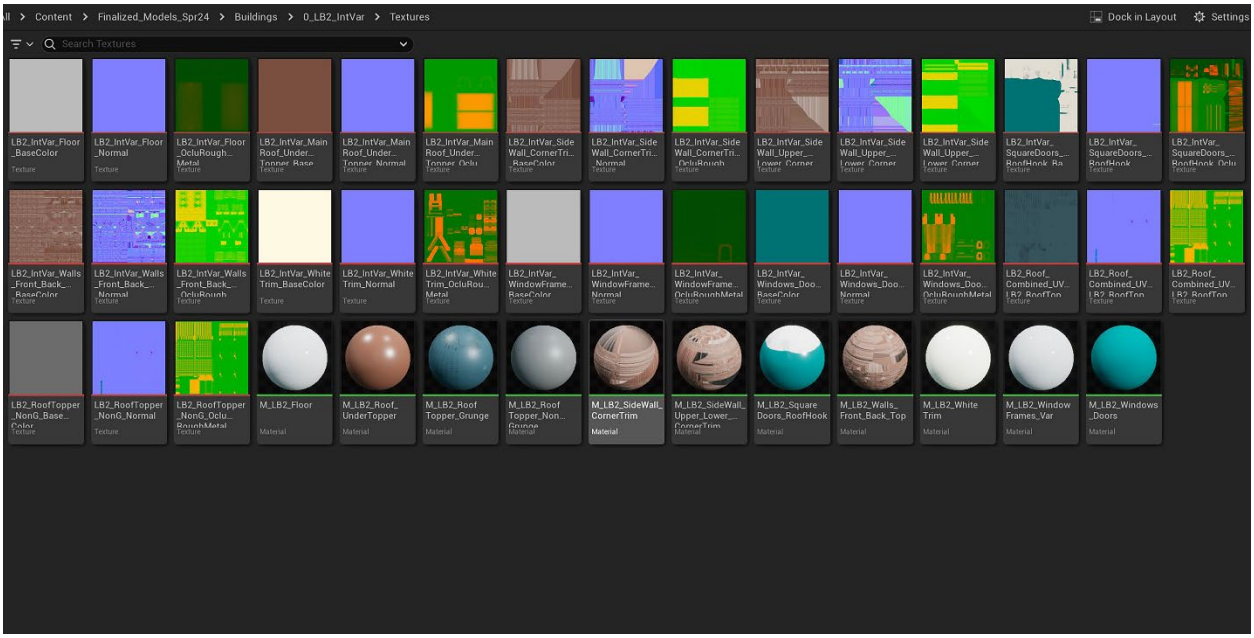

Figure 9: Asset Amount Showcase for LODs and Nanite: Textures and Materials

Figure 10: Model, Texture, and Material set up for LOD Levels 0-2: Front View

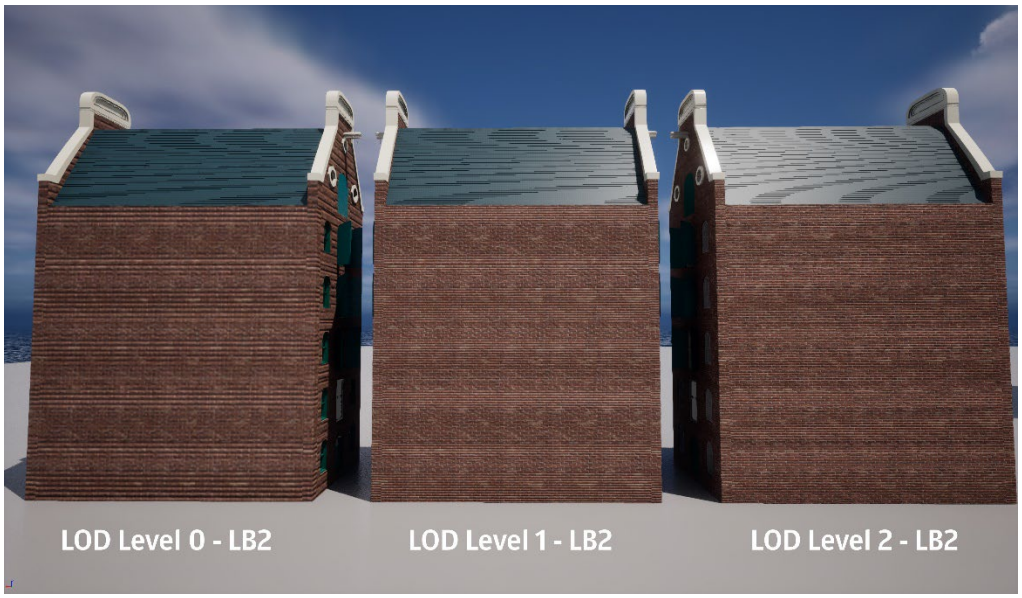

Figure 11: Model, Texture, and Material set up for LOD Levels 0-2: Side View

For the second topic, when implementing Nanite on an imported static mesh, Unreal Engine will begin a process where it breaks up the model's topology into small clusters containing 128 triangles that, when combined, will make the automatic LOD that the engine uses to lower and raise the given model's viewable polygon count when viewing in close or far away distance by the player respectively. However, as seen in Figures 12 and 13, if Unreal does not have a properly smoothed polygon model to use in its cluster calculation for building Nanite on a given model, the process can produce undesirable results in the final calculation. If the windows on the right building in Figures 12 and 13 are examined, the issue seen is the one described above due to the initial placeholder building using a LOD level 0 mesh compared to the finalized building on the left, which uses LOD Level 2 or smoothness level showcased in Figure 7. In addition, if the camera or player's view is brought closer to both models, as seen in Figure 14, then the abnormal artifact disappears due to the closer cluster calculation not needing to change the initial topology to stretch to make the desirable LOD state for closer viewing by the player. However, the artifacts are still present due to their shadows still being calculated and cast across the building model, which cannot be fixed within this Nanite mesh, no matter the level of LOD used for the visual calculation and showcase. With these examples, it is crucial to ensure that if using Nanite on a given static mesh, that developer utilizes either a LOD 1 or 2 smoothed model, seen in Figure 7, to ensure that when the cluster and rebuilding process occurs for a given static mesh, Unreal has enough usable topology to build the varying automatic LOD states needed for Nanite to properly visualize the static mesh from the different distances required for use within game environments.

Figure 12: Nanite Showcase with High and Low Topology Far Camera View

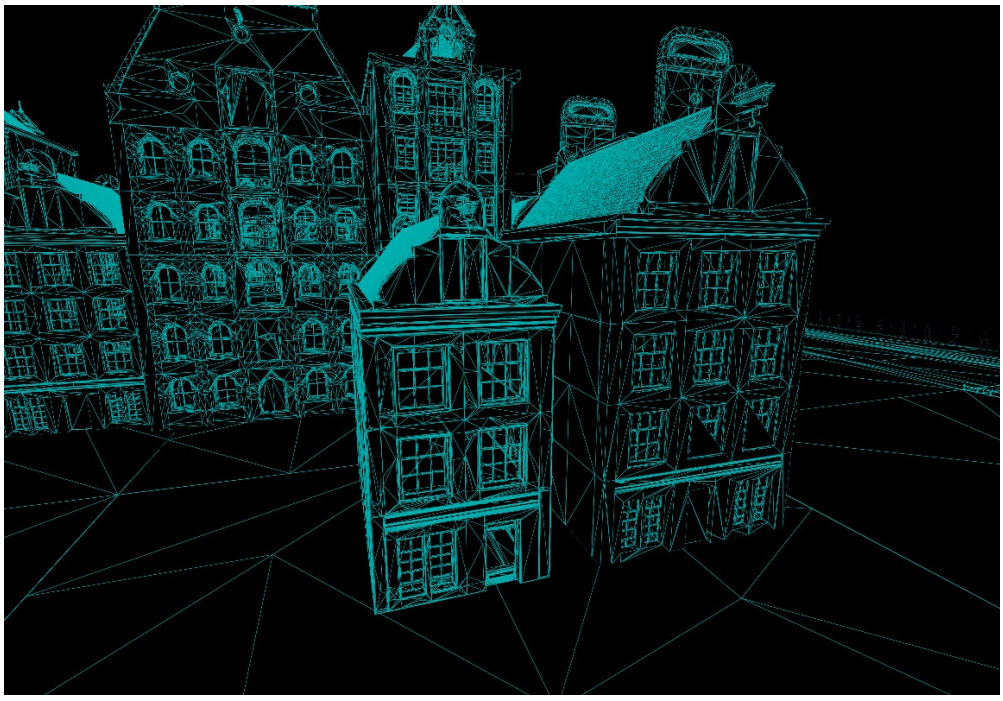

Figure 13: Nanite Showcase with High and Low Topology Far Camera View Wireframe

Figure 14: Nanite Showcase with High and Low Topology Close Camera View

For the final point, using Nanite allows developers to create automatic LODs for their finalized assets, allowing quicker creation, integration, and usage within their given environments. However, there are current limitations with Nanite that developers should avoid when creating and working with environmental assets during their creation and implementation into an Unreal Engine 5 environment. For example, when implementing the exterior streetlamps using Nanite, as seen in Figure 15, the lamp shade section was unable to be assigned to a transparent material such as glass, while using Nanite, due to the process not currently supporting non-opaque materials that use transparent textures, double-sided textures, or texture masks. After this issue was discovered, the devised solution was to separate and import the different sections of the model, the lamp body and shade, as unique static meshes. From there, the lamp body was converted into a Nanite mesh, just as the previous process was used for all

other models; however, the lamp shade static mesh was left as a normal LOD-based static mesh to allow for the application of a transparent glass material. After the static meshes were finalized, the finalized lamp prop was created, as seen in Figure 16, by taking the Nanite enable lamp body and textures, the LOD static mesh lamp shade and transparent glass material, and a point light source actor. In creating this combination, the initial concept for the lamp was finalized, as seen in Figure 16, by overcoming the initial Nanite limitation with the use of LOD creation practices mixed with Nanite to create an asset that took advantage of benefits from both processes without sacrificing the initial concept for this given environment prop.

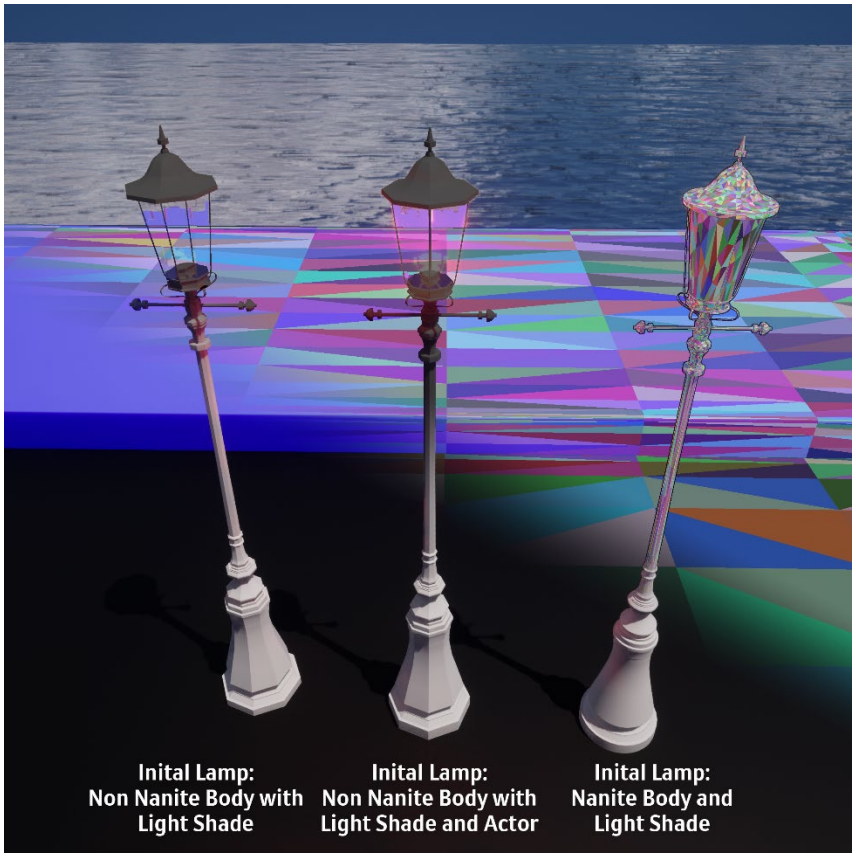

Figure 15: Initial Lamp Prop – Testing Nanite with Non-Opaque Materials

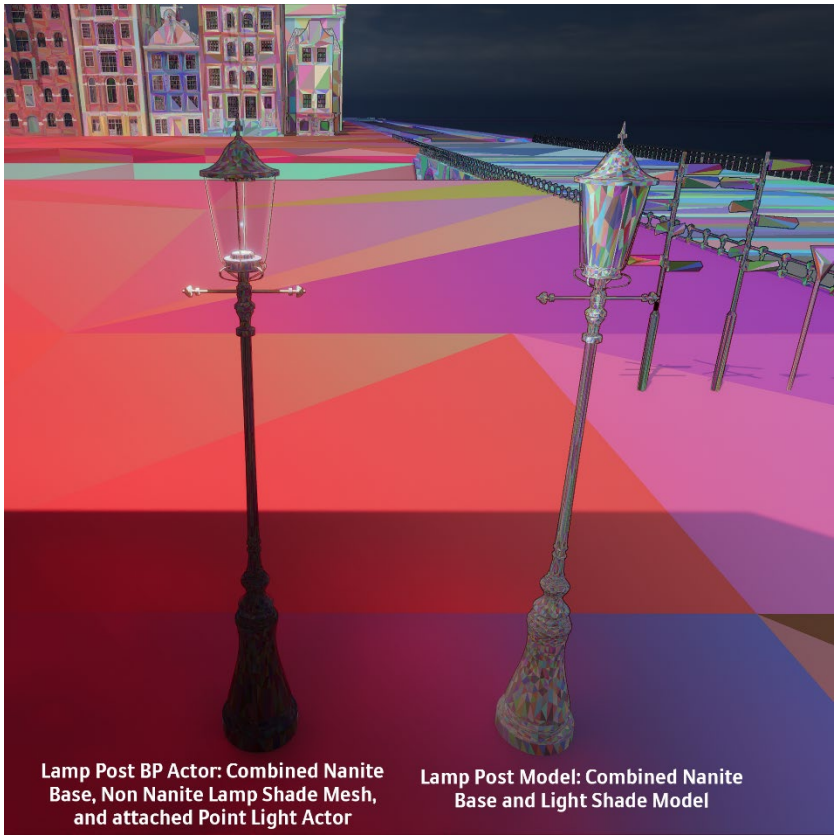

Figure 16: Finalized Lamp Prop – Showcasing Initial and Final Implementation

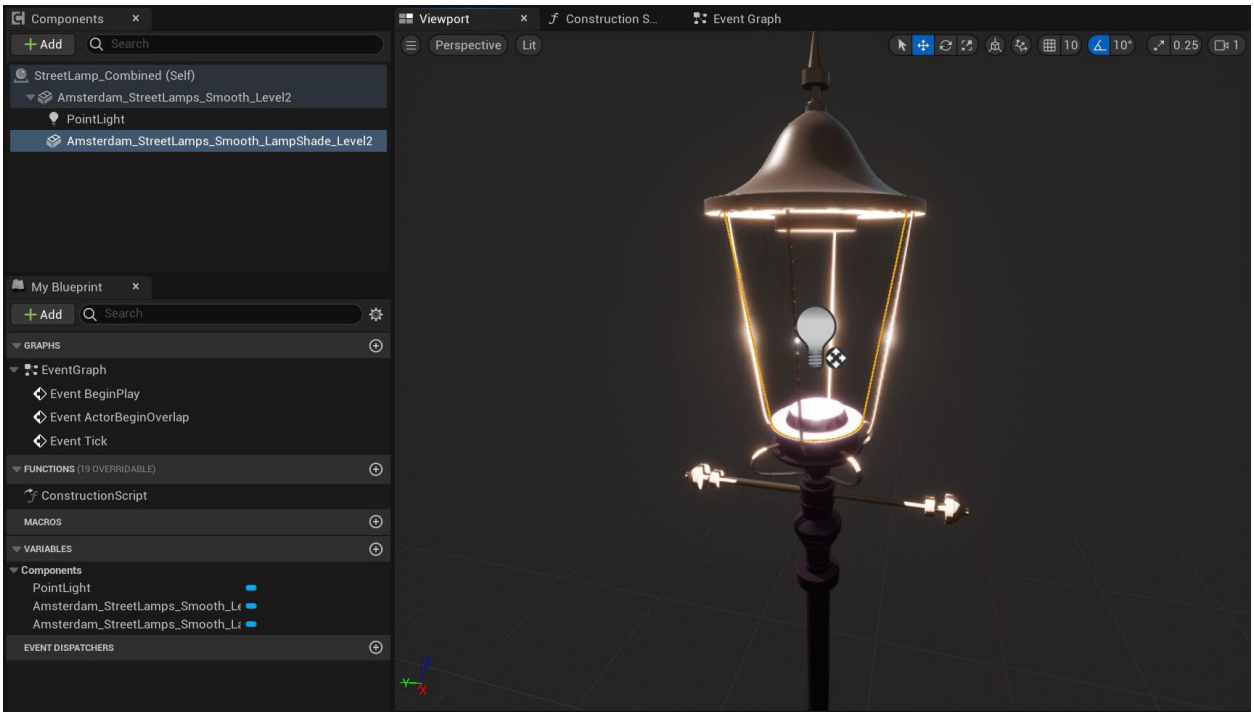

Figure 17: Finalized Lamp Actor BP using Nanite, Glass Material, and Point Light Source

After reviewing the examples above, using Nanite benefits asset creators by cutting the workload by over half of what is required to create, integrate, and implement assets into Unreal Engine 5 using traditional LOD methods. However, as seen with these same examples, there are currently issues with using Nanite inside of Unreal Engine 5, such as abnormal automatic LOD calculations or the support of non-opaque materials that developers will have to understand, workaround, and integrate into their pipelines when developing assets for use with Nanite. Even with these current limitations, the benefits of Nanite help to counteract the potential issues present when using Nanite in the most recent versions of Unreal Engine 5, and as future versions of the engine are produced, the cons could become completely obsolete, which would allow for the full use of Nanite in any developer's pipeline for asset creation. With these points made, Nanite's uses in asset creation are vast and present a leap forward in how assets are created when compared to the older methods of LODs; however, as previously discussed, completely replacing LODs with Nanite is only partially feasible at the time of this research project. Until these limitations become resolved in future versions of Unreal Engine 5, developers can still utilize the benefits and best practices from both LODs and Nanite that aid them in their given creation pipelines to create game environments that take advantage of emerging processes in developing 3D game assets.

Chapter 5. Results

With the research, development, and testing of the modular asset kit wrapping up, the final section of the project was to create the finalized environment for showcasing Nanite and LOD together within the same environment and analyze the results gathered from the final implementation of the modular asset kit. When creating the showcase environment, as seen in Figure 18, the design and layout were designed to roughly mimic a 1-mile section of the city of Amsterdam with identifiable aspects associated with the city, such as its waterway canals, residential and shopping areas, and unique building architecture, while also highlighting the use of Nanite and LODs within the finalized modular asset kit. In addition, as seen in Figure 19, viewers can see how Nanite impacts all assets created in the finalized modular asset pact, where the result incorporates either Nanite or a mixture of Nanite and LOD-based 3D assets. With the completion of the final showcase environment, using Nanite and LODs helped to provide and reinforce the key ideas and aspects learned during this project's research and testing phases.

After this showcase of the environment, the finalized results gathered from both the testing and showcasing phases showed the use of Nanite to be overall positive in creating modular asset kits in multiple ways. First, developers can utilize existing LOD-based creation pipelines to create 3D models and textures with Nanite. In implementing this point, developers do not need to invest as much time or resources to develop Nanite variations of their modular asset kits, which allows these extra resources to be used in expanding their given asset kits in terms of overall details and functionality within Unreal Engine 5. Second, developers can overcome the currently unsupported aspects presented by Nanite by substituting traditional LOD-based methods, thus not impacting developers who wish to use aspects unsupported in current versions of Unreal Engine 5. This point would allow developers to pick and choose when and where Nanite is used,

which can be beneficial if said asset uses things not currently supported by Nanite, such as non-opaque materials or needs if developers need the finalized asset to perform actions that are better suited for LOD-based assets. Lastly, asset kits developed with Nanite in mind can reduce the work required to create asset kits compared to LODs while allowing developers the option to develop LOD kits if needed for a given platform's requirements. When implementing this point, developers can still create LOD-based assets for the platforms that do not support Nanite and implement Nanite for those platforms that do support the process without creating separate unique assets or pipelines that would cause extra work for developers.

After reviewing these results, it is clear that Nanite is a process all asset creation, environment, and Unreal artists and developers need to learn, experiment with, and utilize in their future Unreal Engine 5 projects not only to help cut down on system and development resources but also to cut down on the repetitiveness that comes from using LOD based creation methods to allow developers to place more time in creating higher detailed models and textures that would not be possible if using LOD methods alone. In addition, the current research from the development of this study's modular asset kit points to the current benefits and limitations of Nanite in the version of Unreal Engine 5 used for this study, UE 5.21, and these benefits and imitations will change and update as the process if refined in future iterations of Unreal Engine 5. With this said, this study's research and points will eventually need to be re-evaluated and tested to see what points hold up from the initial research, testing, and implementation and what parts have been improved upon that help bridge the gap between using solely Nanite instead of a mixture of Nanite and LODs in modular asset kit creation for use in Unreal Engine 5.

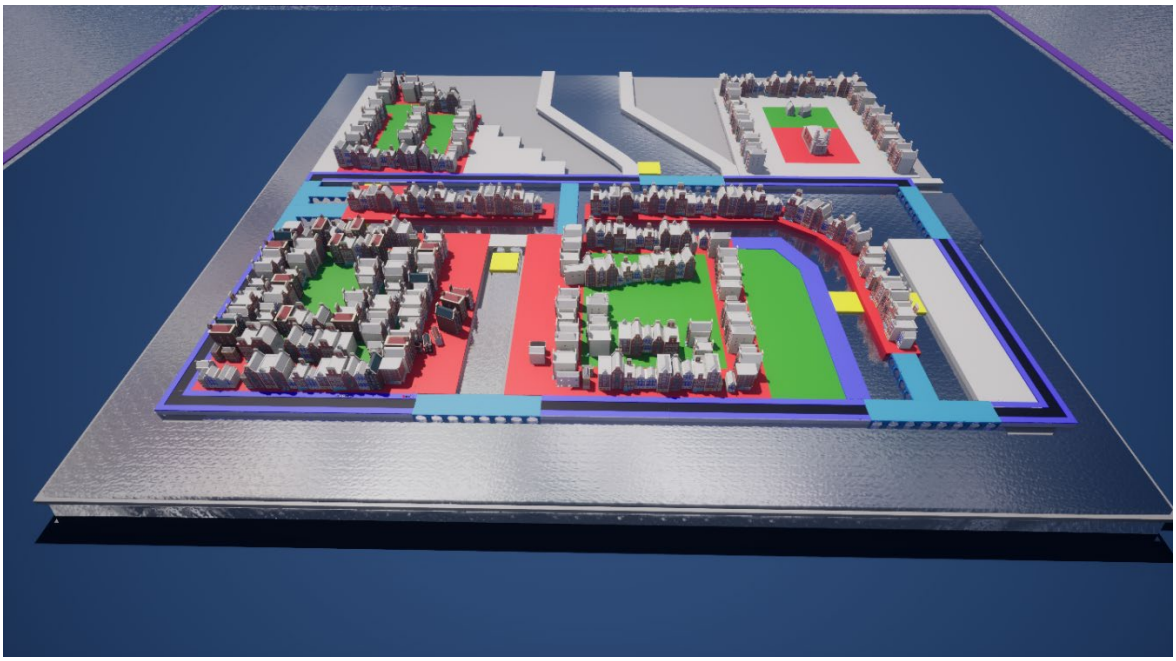

Figure 18: Modular Asset Kit Environment In-Development Showcase in UE5 – Shaded View

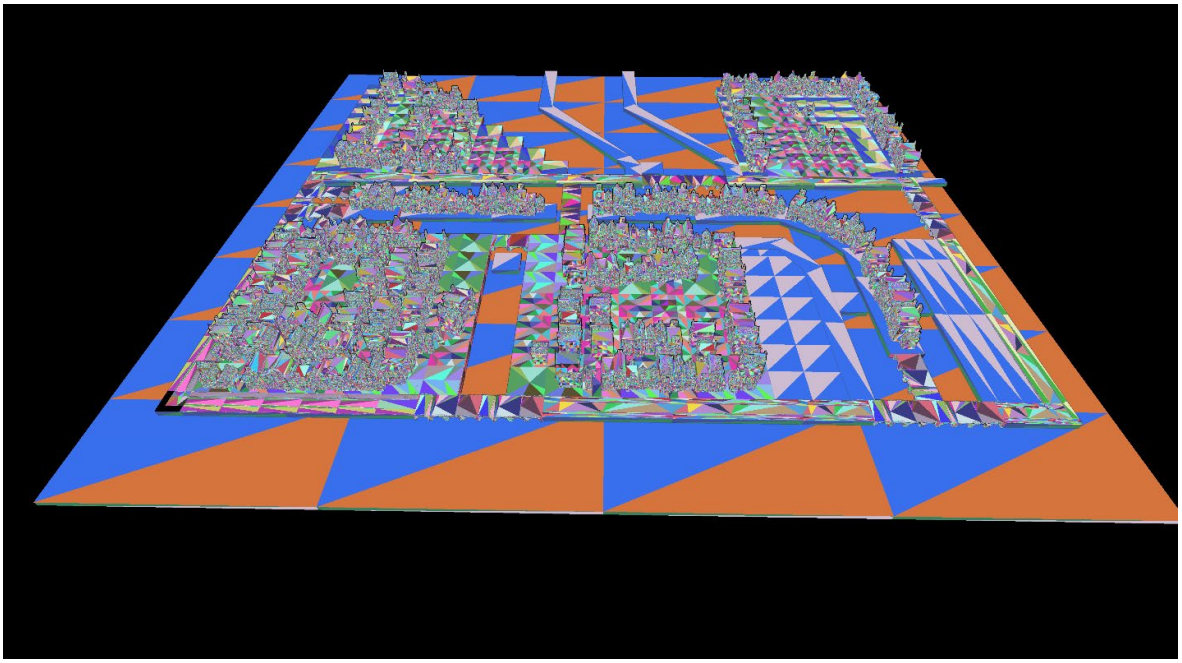

Figure 19: Modular Asset Kit Environment In-Development Showcase in UE5 – Nanite View

Chapter 6. Conclusion & Future Research

Throughout the gaming industry, the workflow of creating LODs has been utilized to create 3D assets for use within game engines. This process has allowed developers to build large-scale game environments while adhering to and overcoming technological limitations that would otherwise limit the assets developers can implement within their environments. However, with the release of Unreal Engine 5, the automatic LOD creation process, Nanite, was introduced to help developers reduce the work required of traditional LODs by automatically creating LOD model variations during gameplay rather than having to implement multiple separate unique models and textures that come with LOD creation methods. In addition, Nanite allows for processing higher-polygon 3D models during the Nanite generation for each 3D model, enabling developers to reduce the resources needed to create highly detailed 3D assets without sacrificing the requirements and system limitations that must be kept in check when creating modular assets kits for use within game engines. With these innovations using Nanite, the question of whether Nanite would replace LODs as the dominant workflow when creating 3D assets for use within Unreal Engine 5 arose. Thus, a study was conducted to test this question by researching LODs and how they impacted assets created for Nanite. After performing the initial research into both LODs and Nanite, a modular asset kit for specific use with LODs and Nanite was created to test both processes when implementing the kit's assets and develop the finalized environment to showcase the process. After testing the kit's asset with both LODs and Nanite, the conclusion discovered is that Nanite can help developers in creating their modular asset kits, but due to current limitations when using Nanite, it cannot at this time fully replace traditional LODs in the asset creation pipeline. However, even with Nanite's current limitations, developers can benefit from both processes by utilizing Nanite to get automatic LOD generation during gameplay and

LODs when Nanite's limitations need to be overcome to get the intended results from the given 3D asset. In future versions of Unreal Engine 5, the limitations associated with Nanite will either be improved upon or removed altogether, requiring future studies and research into using Nanite in Unreal Engine 5, which will lead to my future studies of this given topic.

For future research and testing with Nanite inside of Unreal Engine 5, I plan to investigate more current advancements made with Nanite inside of UE 5.3.2, which came out during the middle of this project's production, to experiment with the new features of Nanite such as its uses with landscape and foliage creation. In experimenting with these new processes and techniques, I would continue to either use and expand upon the currently created modular asset kit from this project's study, or I would make a new modular asset kit to better take advantage of both the newer process and the techniques I learn during the creation of this project's kit. In addition to these new processes in UE 5.3.2, I plan to continue experimenting with the concept of Nanite and LODs by creating new modular asset kits that continue to push the limits of both processes to help expand the knowledge and understanding of those use both processes together in future iterations of Unreal Engine 5. Lastly, I plan to start integrating different modeling processes into my experimentation with Nanite in future versions of Unreal Engine 5, such as procedural or photogrammetry-based modeling, to understand the benefits and limitations of Nanite outside of LOD-based modeling methods.

References

McRoberts, D. A. K., & Hardy, A. (2007, October 1). *Level of Detail for Terrain Geometry Images*. ACM Digital Library. Retrieved March 5, 2023, from https://dl.acm.org/doi/10.1145/1294685.1294689.

Nanite Virtualized Geometry. Unreal Engine | Unreal Engine 5.0 Documentation. (2022). Retrieved March 5, 2023, from https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/.

NewsRX LLC. (2022, April 19). *Methods and Systems for Generating Level of Detail Visual Assets in a Video Game*. Gale General OneFile. Retrieved March 6, 2023, from https://link.gale.com/apps/doc/A700741702/ITOF?u=tel_a_etsul&sid=bookmark-ITOF&xid=cc4ed738.

SIGGRAPH Advances in Real-Time Rendering. (2021, October 29). *A Deep Dive into Nanite Virtualized Geometry* [Video]. YouTube. https://www.youtube.com/watch?v=eviSykqSUUw.

Unreal Engine. (2022, November 3). *Building Open Worlds in Unreal Engine 5 | Unreal Fest 2022* [Video]. YouTube. https://www.youtube.com/watch?v=EEf07ggFWRw.

Unreal Engine. (2022 April 5). *The Matrix Awakens: Creating a World | Tech Talk | State of Unreal 2022* [Video]. YouTube. https://www.youtube.com/watch?v=xLVJP-o0g28.

Unreal Engine. (2022 April 5). *The Matrix Awakens: Generating a World | Tech Talk | State of Unreal 2022* [Video]. YouTube. https://www.youtube.com/watch?v=usJrcwN6T4I.

Unreal Engine. (2021, August 3). *Nanite in UE5: The End of Polycounts? | Unreal Engine* [Video]. YouTube. https://www.youtube.com/watch?v=xUUSsXswyZM.

World Partition - Hierarchical Level of Detail. Unreal Engine | Unreal Engine 5.0 Documentation. (2022). Retrieved March 5, 2023, from https://docs.unrealengine.com/5.0/en-US/world-partition---hierarchical-level-of-detail-in-unreal-engine/.

Zhong, Y., Yun, T. S., & Lee, B. C. (2021, December). *The workflow of making realistic 3D model by combining photogrammetry and nanite*. DBpia. Retrieved March 5, 2023, from https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE11025317.

APPENDIX: PROFESSIONAL TERMINOLOGY

1. 3D model/asset/mesh – A 3D object created to be used in external programs such as game engines, animation software, etc.
2. Actors – A Blueprint system in Unreal Engine 5 that has proprieties such as Static Meshes and Light Sources added to it for easier placement during development.
3. Asset – A exported object whether 3D model, textures, etc. for use within different 3D programs.
4. Level of Detail (LODs) – A process by which developers use lower quality instances of static meshes to render out high or low visual fidelity depending on player distance to an object.
5. Materials – A finalized texture, comprised of multiple texture maps or layers, in Unreal Engine 5 or Substance 3D Painter, respectively.
6. Modular Asset Kit – A 3D development asset pack that is filled with modular 3D models, textures, etc. for use in 3D software such as Autodesk Maya or Unreal Engine 5.
7. Nanite – A process by which UE5 uses a base static mesh to constantly calculate the visual fidelity of a mesh based upon player distance and visibility of parts of the mesh to the player.
8. Platform – A reference to the hardware that will run video game software once development is finished.
9. Static Mesh – A 3D model that has been integrated/imported for use inside of Unreal Engine 4/5.

VITA

Stephen R. Overton

Education: M.F.A Digital Media, East Tennessee State University, Johnson City, Tennessee, 2024

B.S. Digital Media, East Tennessee State University, Johnson City, Tennessee, 2021

Professional Experience: Graduate Associate/Lecturer, East Tennessee State University, College of Business and Technology, Johnson City, Tennessee, 2021-2024

Publications: Guest Speaker at East Coast Gaming Conference 2022. “Lessons learned in VR training partnerships East Tennessee Children’s Hospital and ETSU.”

Honors and Awards: American Advertising Awards Online/Interactive Social Media Single Execution – Silver Addy Award (Student), AAF Northeast Tennessee, 2022

American Advertising Awards Elements of Advertising Art Direction– Single – Gold Addy Award (Student), AAF Northeast Tennessee, 2022

American Advertising Awards Elements of Advertising – Film, Video & Sound – Animation or Special Effects Gold Addy Award (Student), AAF Northeast Tennessee, 2022

American Advertising Awards Best of Show (Student), AAF Northeast Tennessee, 2022