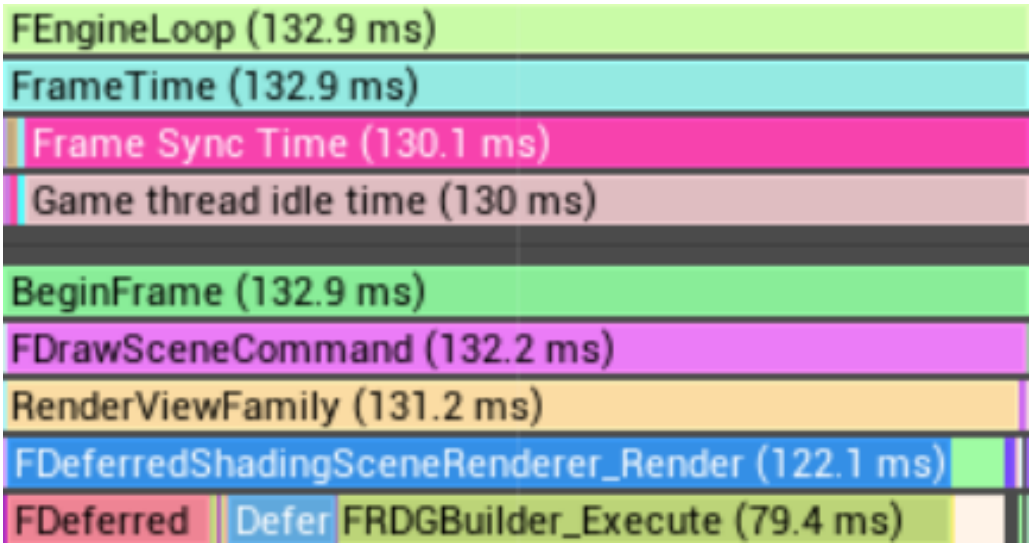
RPG NA - Optimisation Document

Identification Methods

- Web resources used for documentation on optimizing in Unreal:
  - Unreal Engine Game Optimization on a Budget
  - Unreal Engine 4.27 - View Modes
  - Unreal Engine 4.27 - Building Texture Streaming Data
- `stat detailed` is the most useful command for quick profiling. It shows FPS, frame times, and a frame graph.
- General stress points are identified by playing in the editor.
- Once one is found, a build is made for in-depth analysis.
- For specific investigation, debug view modes are useful: light complexity, lightmap density, shader complexity, quad overdraw, mesh UV density.
- Unreal Insights is the go-to tool to find bottlenecks, as well as precise values for individual events in a frame.

Original State of the Project

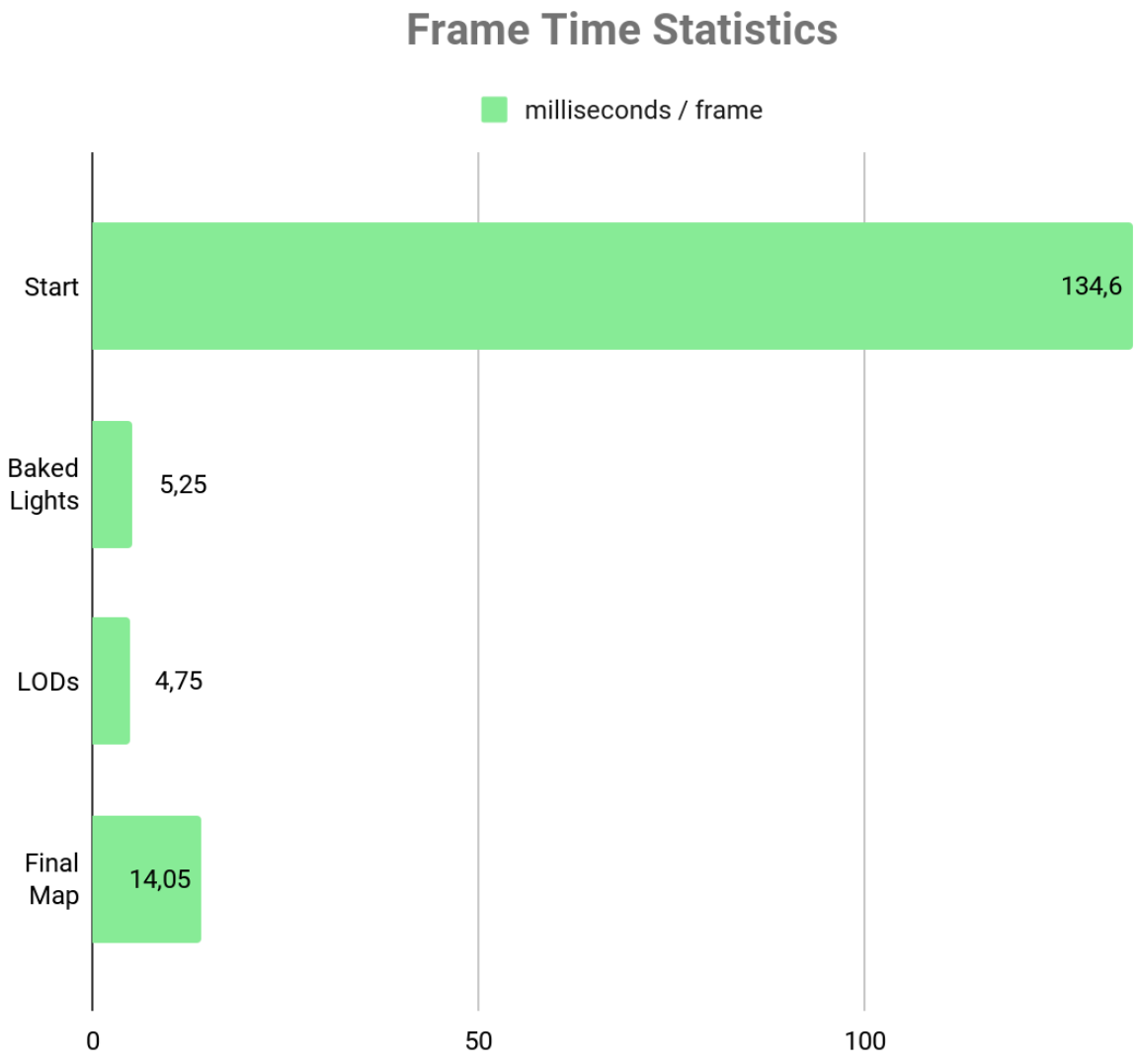
- Game is GPU bound, with the *FDeferredShadingSceneRenderer_Render* shader pass taking up most of the frame time.
- Stats are: 7.4FPS and 134ms frame time (with 124ms shader pass).



Unreal Insights session on a build of the game with no optimizations.

Overview of Improvements

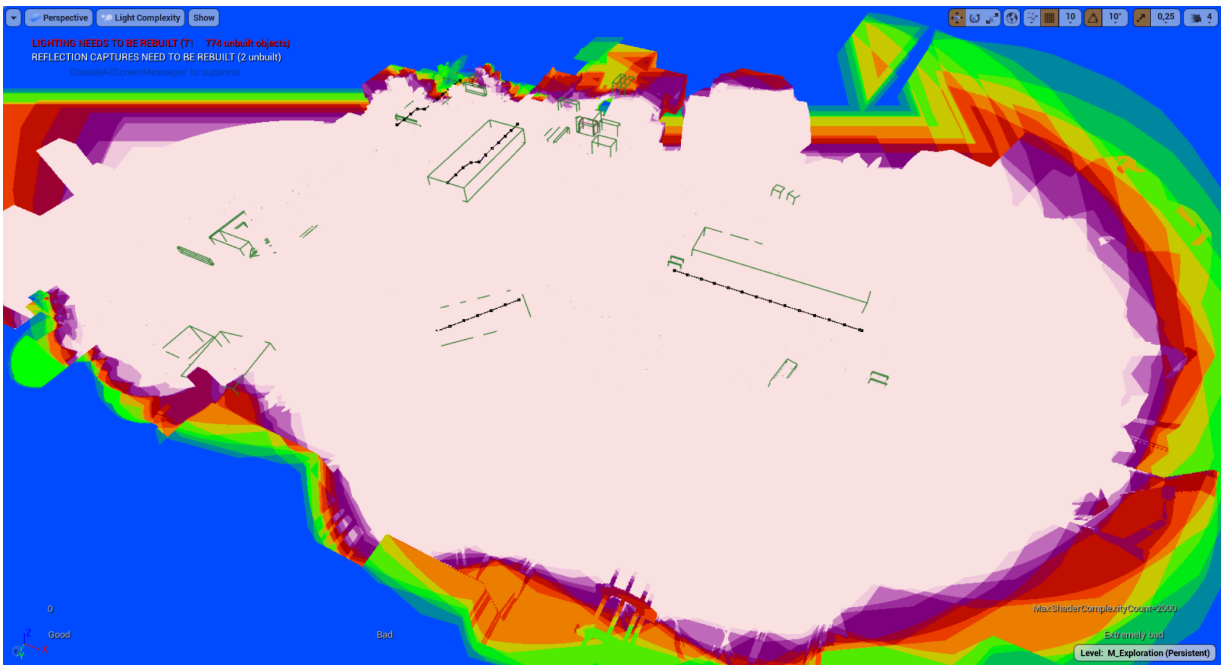
- The following graph shows a list of the successive optimizations we made to the project.
- The bars show the average time per frame at each step of the process.
- Each entry improves on the performance of the previous one.
- Key values are 33ms for 30FPS, 16ms for 60FPS, and 6.95ms for 144FPS.



Graphics Case

Extremely bad light complexity

- Identification: Light complexity viewmode shows values for most of the scene in the “Extremely bad” category.
- Resolution: Set all lights to static and bake the scene. This causes a problem however, since static lights don’t cast shadows on dynamic actors such as the player. This means it is necessary to pick which lights to set to static, and which to keep as stationary.
- Result: 190FPS and 5.25ms frame time.

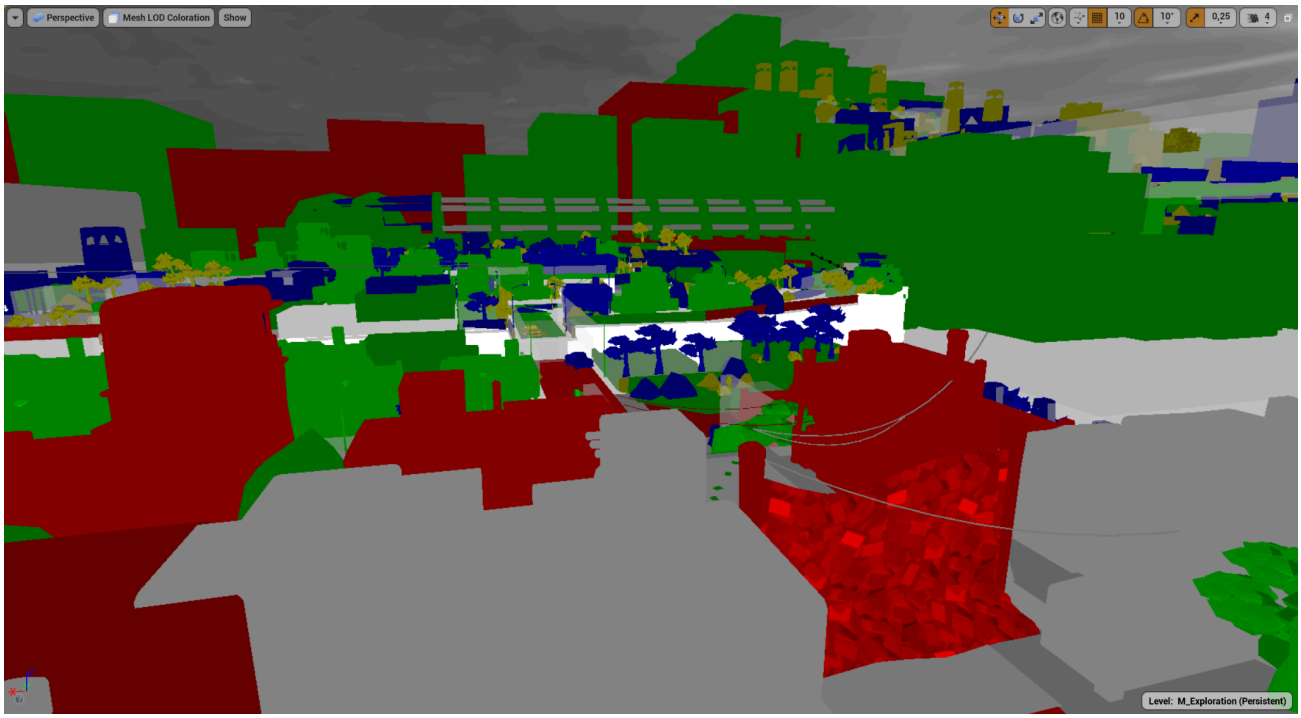


Exploration map with unbuilt lighting in light complexity viewmode.
(blue: good, white: extremely bad)

Many complex meshes on screen at once

Levels of Detail

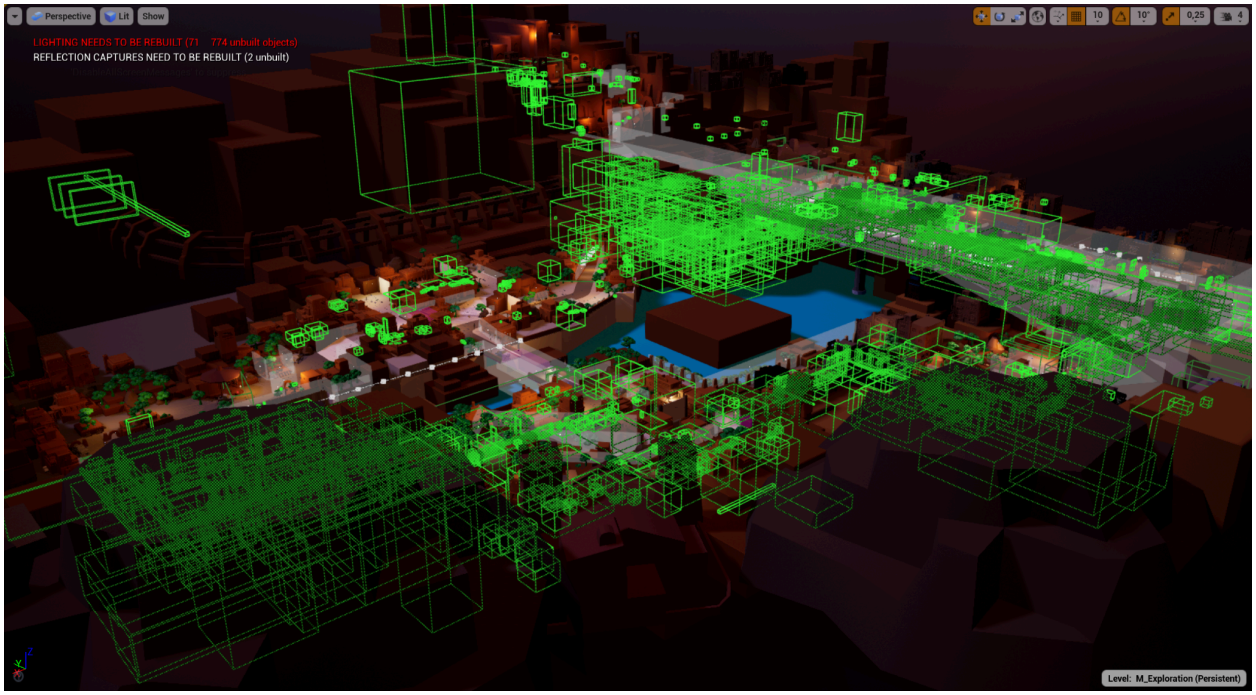
- LODs can be generated by the engine for any static mesh. This makes it possible to render a huge amount of assets at once, while keeping the vertex count low enough.
- Resolution: Create an asset action blueprint that makes it possible to right-click any StaticMesh and select “Generate LODs” to automatically set up 5 levels of detail. There are 3 quality settings: Low, Medium and High. Currently, all meshes in the project are set up on Medium, but discussion with the artists will be necessary to choose the right quality for each type of mesh.
- Result: 210FPS and 4.75ms frame time.



Exploration map in LOD coloration viewmode.
(gray: original quality, red: lod1, green: lod2, blue lod3, yellow: lod4)

Culling and Instancing

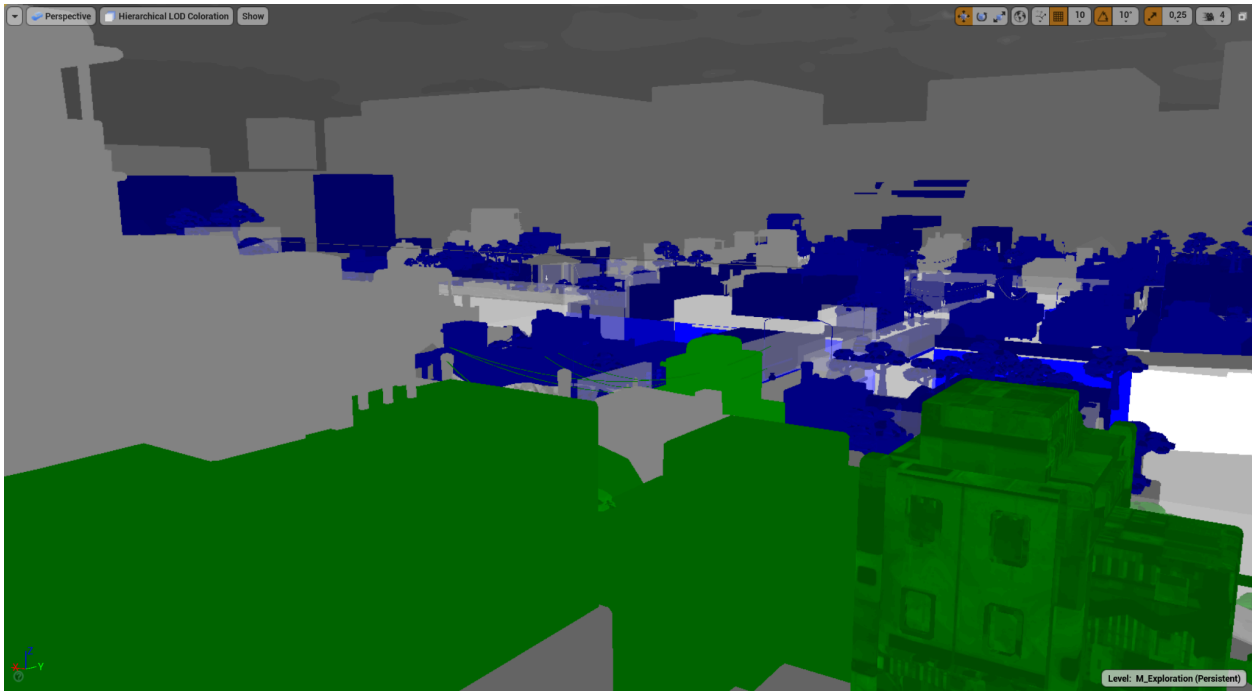
- Frustum Culling, Occlusion Culling and Mesh Instancing help tremendously in this situation, and they are all enabled by default in Unreal Engine.
- Distance Culling, however, isn't enabled by default. To enable it, we placed a cull distance volume around the whole level and set up multiple distances at which actors of different sizes should be culled.
- To optimize Occlusion Culling further, static meshes can be assigned a specific LOD for occlusion testing. By default, this is set to 0 (max quality), but we changed it to level 3 for all meshes.
- Commands used:
  - `r.visualizeoccludedprimitives 1`
  - `stat initviews`
  - `stat rhi`



Exploration map from the vista at the end of the tutorial.
Green boxes show all the meshes that are occlusion culled.

Hierarchical Levels of Detail

- To optimize culling and draw calls even further, HLOD actors can be generated to group meshes together, creating proxy meshes. This way, they can be culled together in a single check, or drawn together in a single call.
- Resolution: Enable Hierarchical LOD System in World Settings, then generate HLOD clusters and proxy meshes.
- Problems:
    - Many actors were not included in the HLOD system, as can be seen in the picture below.
    - Actors included in the HLOD system were never distance culled and always rendered at LOD 0 (original mesh quality), because only that level was baked into the proxy meshes.
- Result: Marginal improvement because of the problems mentioned above. They were either the result of bugs in the HLOD system (it was improved in UE5), or my potential lack of understanding of the feature. In the final build, Hierarchical Levels of Detail are disabled.



Exploration map with HLODs enabled.
(gray: no HLOD, green: HLOD 0, blue: HLOD 1)

Level Streaming

- Level Streaming is also a great way to keep the memory overhead low, by unloading far away or not visible parts of the level. However, to implement this, it would be necessary to section the current scene into multiple levels, which would be very time consuming. Since the performance is sufficient and the level designers are crunching, we decided against implementing this optimization.

Problematic actor mobility

- The mobility for many meshes (mainly foliage) and lights was set to “Movable”. As more and more were added to the exploration level, performance dropped steadily on both CPU and GPU sides.
- Resolution: Make an ActorActionUtility script to set all selected static mesh actors or lights to “Static” mobility.

Cascaded shadow maps

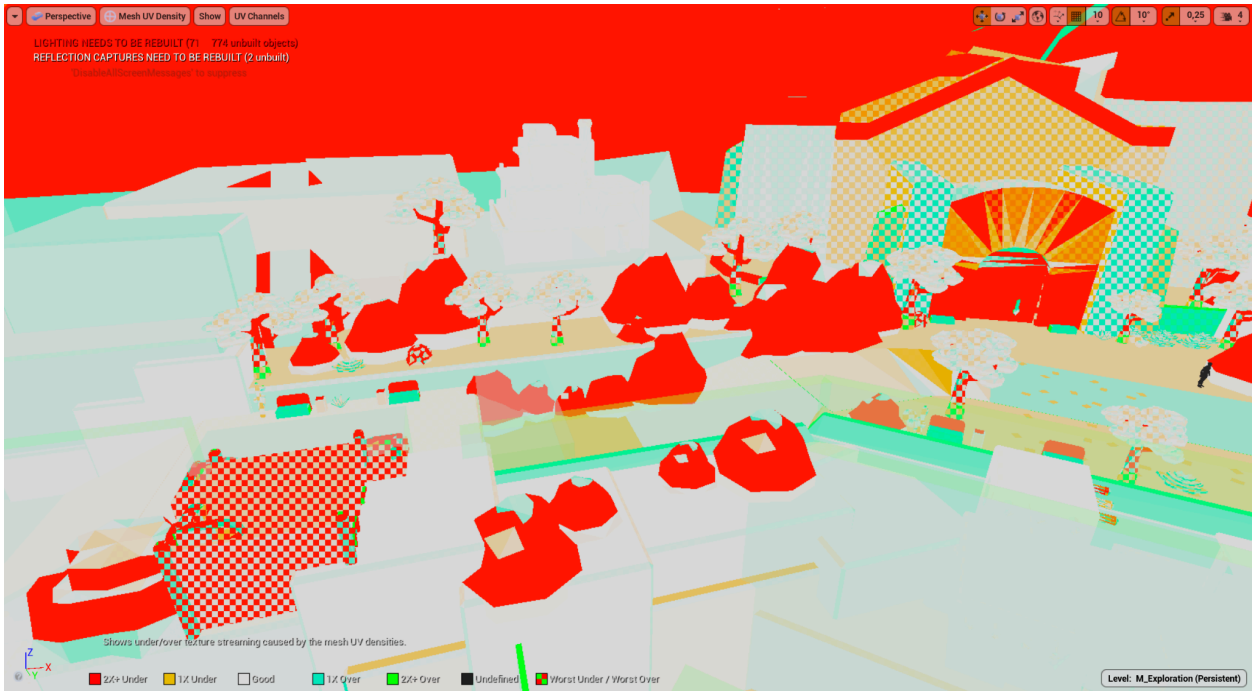
- To squeeze a little bit more performance out of dynamic shadows from directional lights, cascaded shadow maps can be enabled. This reduces the quality of faraway shadows to save on GPU computations, while still showing very high-quality shadows up close.
- Resolution: Select a directional light and set appropriate settings in the “Cascaded Shadow Maps” section of the details panel. The light must have its mobility set to “Stationary” or “Movable”.

Precomputed Visibility

- In a game that uses many camera tracks, actor visibility can be precomputed for frustum and occlusion culling. This saves the render thread some work, while slightly increasing runtime memory usage and level build times.
- Resolution: In a scene’s world settings, enable “Precompute Visibility” and choose an aggressivity level.

Bad mesh UV density

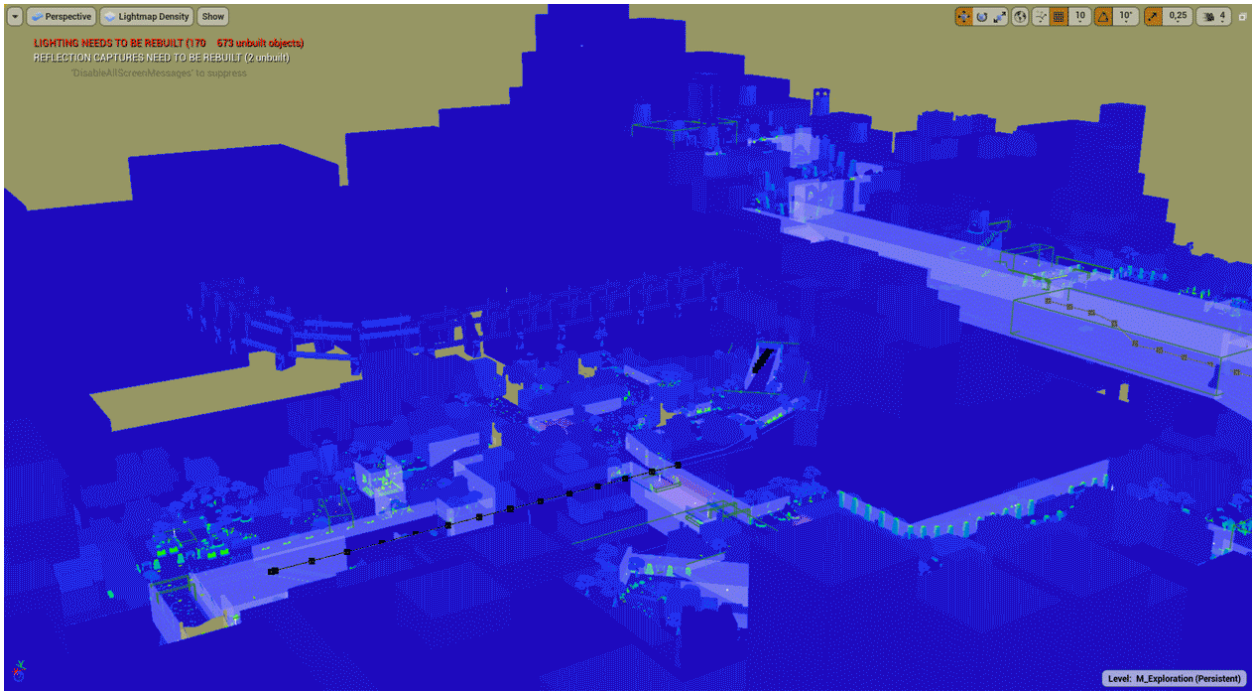
- Identification: The mesh UV density viewmode shows values for many temporary meshes in the “worst over/under” and “2x+ under” categories.
- This causes some performance issues, as well as problems with Lightmap Density when building the scene’s lighting.
- Should be fixed with new final version meshes.



Exploration map in mesh UV density viewmode.
(cross-hatch: worst under/over, red: 2x+ under)

Sub-optimal lightmap density

- Identification: Lightmap Density viewmode shows values for most of the scene in the “Less than ideal texel density” category, which means the quality of baked lights is sub-optimal.
- Resolution: Set the overriden lightmap resolution to an appropriate value for each static mesh component in the scene. To automate this process, we found the AutoLightmapAdjuster plugin, which we modified to get most scene objects into the “Ideal texel density” category.



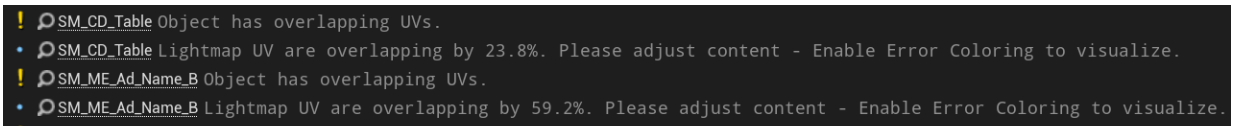
Exploration map before and after the execution of the plugin.
(blue: less than ideal density, green: ideal density, red: greater than ideal density)

Many textures don’t have an optimal size

- Textures should be square and have a size that is a power of 2 for best compatibility and performance.
- Most UI textures have arbitrary size, which could cause performance issues if many were present on the screen at once. However, this isn’t the case, and most mesh textures have an optimal size, which is the most important.

Overlaps in texture coordinates

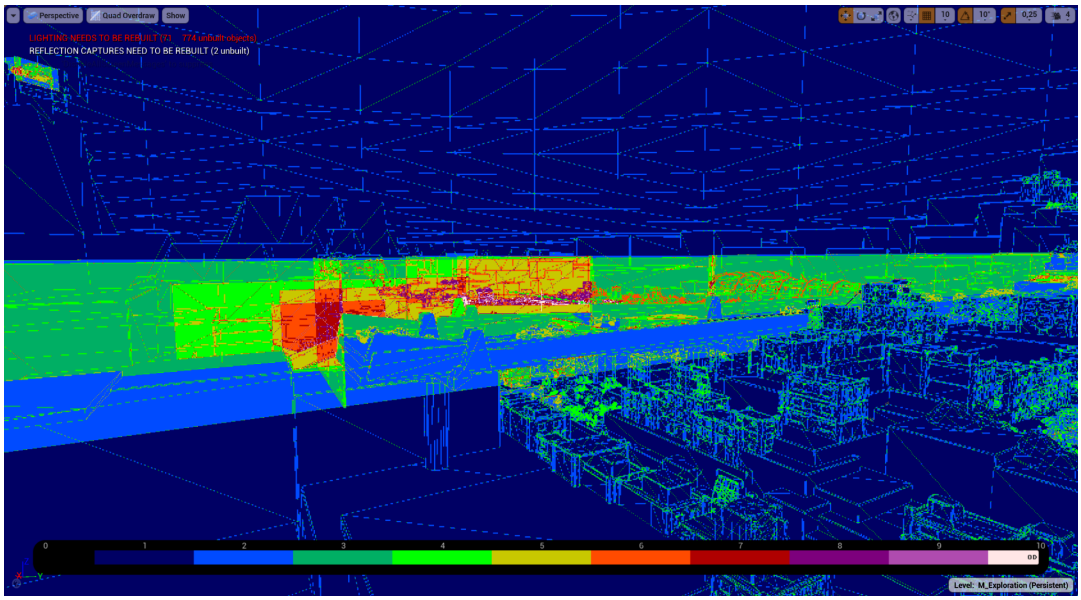
- Identification: Building the scene prompts many warning logs about UVs overlapping in many temporary meshes.
- This should be fixed with new final version meshes.



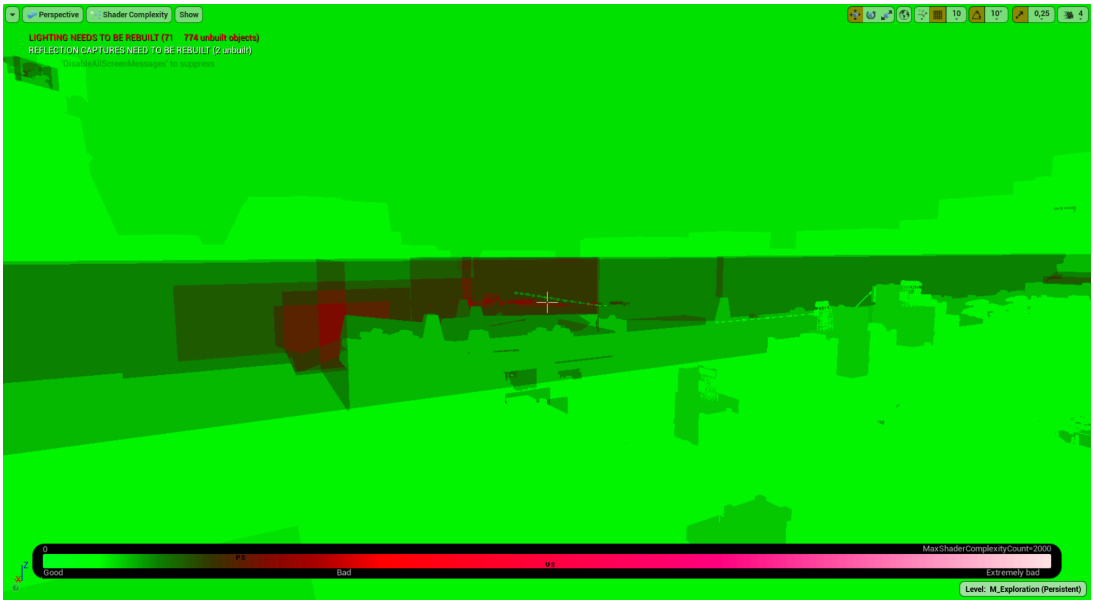
Message logs for lighting build results

Quad overdraw & Shader complexity

- Identification: The quad overdraw viewmode shows a few values at or above 10 overdraws. Similarly, the shader complexity viewmode highlights some problematic places.
- This is a very minor issue as it only happens with transparent objects. Those are sparse in the scene and most aren't rendered at all in game.



Exploration map in quad overdraw viewmode
(dark blue: 1 quad drawn, white: 10+ overdraws)



Exploration map in shader complexity viewmode
(green: good, red: bad, white: extremely bad)

Gameplay Case

Simple game mechanics

- Our project has simple game mechanics that are not very performance hungry. In the exploration scene, the game is still GPU bound, even after the many graphical optimizations we added. TODO for the fight scene.

Converting Blueprint to C++

- The most efficient way to optimize the performance of an Unreal Engine project is to convert slow behaviors from Blueprint to C++.
- One of our GDPs took the initiative to implement potentially slow behaviors in C++, which already helped a lot with performance before we started working on the project.
- To push this further, we could have translated a large portion of the codebase as well, but we found it would be overkill and bug-prone, so we decided against it considering the short deadline.

Optimizing algorithms

- Another way to improve on the performance of Blueprints is to review and optimize the algorithms used for gameplay.
- This is applicable mostly in the fight scene, where the opponent is driven by an AI and random numbers are used intensively.
- This is where most of our coding force was spent, as we improved and optimized the enemy AI, as well as refined the game mechanics offered to the player.

Conclusion

In the end, we mostly worked on graphics optimizations, due to the gameplay's low performance needs, compared to the map's size and level of detail.

We had to go back and forth and debate a lot with artists and designers on lights and LODs to get the best looking result while optimizing for performance as much as possible.

In the end, we came to the agreement that rectangle lights should only be static so as to bring general lighting to parts of the map, while spot and point lights should be stationary only when in range of the player. Finally, the directional light of the moon is stationary, with cascaded shadow mapping enabled for best performance with dynamic shadows all over the map.

LODs, on the other hand, are very aggressive on foliage and decor, while being tuned down a lot for buildings, so as not to break the shape of windows.

To conclude, we managed to keep the game's frames per seconds at a reasonable level, while not sacrificing too much detail. It was a great learning experience that pushed us out of our comfort zone and to new heights as developers.