**Progress Report Timeline**

**(Sudiksha Sharma)**

**---------------------------------------------------------------------------------------**

**---------------------------------------------------------------------------------------**

**Project github : : https://github.com/sudikshasharma/object-inspection-project**

**---------------------------------------------------------------------------------------**

**---------------------------------------------------------------------------------------**

**21 November, 2022 - 2 December, 2022**

**(Project Understanding and Unreal Setup)**

* Get used to of Unreal Engine (get acquainted with basics of UE, scene hierarchy and management, object loading, light setup and building lightmaps,material definition and usage, blueprint for object and scene programming)
* Setup C++ for Unreal

**---------------------------------------------------------------------------------------**

**2 December, 2022 - 23 December, 2022**

**(Setup lights)**

* Tried 2 light setups to replicate the lights of the actual inspection device.
* Setup 1 : Area Lights : Need to extract out the torus shape from rectangle light to mimic the light source of the inspection model which will take extra efforts. (As discussed with Juraj, masking the central pixels in the form of torus using circle equations). Also, there's no prewritten configuration available in UE to do the same. Emissive materials are quicker and give better visual results.
* Setup 2 : Emissive material : Self illuminating materials. Created two 2d torus mesh in Blender with the measurements provided. (Circular Torus and Hexagonal Torus). Created custom emissive material in UE, with emission value (emissive intensity) and rgb range, both defined by user. Turn off eye adaptation (doesn't support real cameras) and other unnecessary properties which take up computational power.

**(Load inspection object mesh)**

* Since mesh was available in .obj file, this file format is NOT fully supported by UE.
* So, I loaded the .obj file in Blender and exported it in fbx format in parts and manually added materials to them in UE.
* Uploaded the project on gitlab in the ‘dev’ branch (link at top).

**(Writing Shader)**

* Understood some important OSL concepts.
* Understood the material editor in Unreal. Unlike OSL and other shader languages, it is visually scripted in the form of connected graphical components. Though, there is scope of writing code as well but graphs are mostly preferred and are quick to use.
* Tried to translate the 2 sample OSL shaders written by Lovro in UE material graphs.
* While translating OSL to Unreal, 2 major problems were faced.
  + In OSL, there are some inbuilt methods like rotate,cellnoise(to add noise). Unreal provides somewhat similar methods but not exactly the same, e.g. some parameters are different which may result in different output. So, the resulting rasterized image is different.
  + Shaders have been made but needs hit and trial with the lighting, scene properties etc. because it is giving out many artifacts at the moment.
  + Strongly requires a way to debug the shader compilation, as there are a large number of nodes, so it's difficult to figure out which node is giving out wrong output and hence tearing the shader. Once, this is being found, then it will be relatively easy and quick to fix and implement all other shaders as well.
* RESULT : The basic cylindrical texture that we want to achieve is there in the render, but it's nowhere near to the renders from OSL. (No noise) There might be a need to tweak some parameters or change some code approach on importing to material graph. Also, there might be a good scope of optimization in the currently implemented material graph.
* Debugged and fixed the shader with Juraj and understood the hacks he discussed to debug the shader.
* The cylinder lines became visible, but a lot of illumination, aliasing issues and unexplained weird artifacts were still there.

—-------------------------------------------------------------------------------

**24 December, 2022 - 26 December, 2022**

**(Shader Verify and Anti-aliasing)**

* Since the shader material translated had a lot of visual defects, so reanalysed the translated shader, used some hacks to debug the shader (as discussed with Juraj)
* As a result, the texture generated had the basic cylindrical impression look, but was far away from the expected output (attached image). It was more of jaggy lines with some unwanted white noise, which is the apparent case of Aliasing and tweaking some light and environment settings.
* Also, there was some issue with textureRotator logic in the material graph (it was rendering texture at a particular rotation value only). This also needs to be taken under consideration after anti-aliasing.
* As a result, Juraj updated the project timeline with anti aliasing on priority.
* Did research on anti-aliasing and its types.
* Types of Anti-Aliasing in Unreal engine:
  + **MSAA (x2,x6,x8)** : It samples 2 or 3 adjacent pixels together at a very high resolution. This smooths out the edges of polygons. This is the main reason why MSAA is much faster than SSAA. It produces one of the best image qualities. This is because it uses edge detection algorithms to detect aliasing then corrects the problem.
  + **TAA :** It uses samples both outside and inside each pixel to deliver high-quality images. Keep in mind that it can be demanding on your hardware.
  + **FXAA :** It does not run calculations on the colors and geometry of a game; instead, it blurs the rough edges. This makes the image a bit blurry and this is not preferred by many.
  + **TSR :** Anti-aliasing is achieved by super sampling the images then rendering them at a higher resolution. From this larger resolution, the method then downsamples the images. The problem with this method is that it requires a huge amount of computing power

—-------------------------------------------------------------------------------

**27 December, 2022 - 5 January, 2023**

* Verified the shader again. Introduced noise parameter as well in the shader- a vector noise (which was missing earlier). As a result, there was a significant positive change from the previous results.
* Looked for ways to change anti-aliasing types and increase the samples in Unreal Engine.
* One way found was in project settings, under the Rendering Section, we can change the anti-aliasing types(4 ways) and sample scale for MSAA(x2,x4,x8).
* Also, increase the engine scalability towards anti-aliasing to the highest (under settings).
* But with current shader implementation and editor settings, there was not much of a difference in the aliasing.
* So, another option available to test eliminate the aliasing was to use level sequencer under cinematics (add plugin) and get a rendered image by increasing the sample count under level sequencer’s anti aliasing settings.
* Default value = 0; Set the value to 10. Outcome : Video memory got exhausted. System got stuck in reboot loop,
* As a result, I had to reset the windows while keeping the personal files but reinstalling all the apps again.

—-------------------------------------------------------------------------------

**5 January, 2023 - 10 January, 2023**

* Discussed the hang issue with Juraj (might be due to getting in a deadlock situation when unreal exhausted gpu on starting rendering at extremely high sample all of a sudden).
* As a result, discarded level sequencer due to being computationally heavy on gpu (it is used to sequence in game cinematics in games) and long time to produce a rendered image.
* Another issue with level sequencers is that we cannot set the camera transforms.
* So, to render the scene at various camera transforms, we must use blueprints. One way to do so is using render target textures.
* But the main problem to resolve for now is to solve the anti-aliasing issue. A few methods were discussed with Juraj to try anti aliasing, along with a few things to take care of :
  + METHOD 1 : Try the above mentioned anti-aliasing methods one by one. Check the global engine settings, e.g. use forward rendering (this is lighter than deferred).Turn of bloom, motion blur and other related properties in engine settings. Use power of 2 in case of sample count (computation of numbers in power of 2 is lighter)
  + METHOD 2 : Shake the camera and capture the render (basically to abuse and use temporal anti aliasing (Reread about this, confusion ? )). Shake in the x-y plane of the camera, within pixel size.
  + METHOD 3 : Texture filtering (Read more about it)
  + METHOD 4 (try not to stoop this low) : Use uv mapping textures instead of procedural texture generation.

* Also, some things to take care of :
  + Set correct resolution for camera(2448, but use 1224, divide resolution by 2 to avoid crashes for now but don't go lower than that)
  + Try forward rendering(cheaper than deferred)
  + Put the screen capture component (which captures the scene like a camera) under camera to render on the render target.
* Result : Anti-aliasing problem was significantly solved by using Method 1 only.
* Comparing the types of anti-aliasing, TSR gave the best results, followed by MSAA (x8).
* Issue faced : Unable to use render texture target to capture the screen render. Due to some faulty settings, the texture was capturing some noise instead of rendering the inspection object. So, I skipped this method to discuss it later with Juraj.
* So, for temporary purposes, just to compare the results of different anti-aliasing methods, I used HighResShot command (to be entered in the command line of unreal) to capture the current render in the editor's view.

—-------------------------------------------------------------------------------

**10 January, 2023 - Current**

* Analyzed the render shots with Juraj. Along with original shader parameters, tried different parameters as well (e.g. changed cylinder distance and rendered) to analyze the render.
* With tweaked parameters, there was a significant difference in shaders, it looked better and somewhat more similar to the original renders (though it shouldn’t be the case as we expected best results with original parameters only. But then, how a method is implemented internally in a library plays a great role in the output, which we can’t control).
* Also, the video memory was getting exhausted very quickly in the editor, so I changed some settings to save some memory, such as : change screen percentage, experimented with realtime on/off. It helped, but not much.
* Discussed the render target issue with Juraj and tried to set it up (adding screen capture component under camera, set render texture size as discussed above and play-stop the scene to capture the render on render texture)
* Along with Juraj, tried different tweaks to get the object rendered on the render texture, such as (Explore below points in detail) :
  + Used LDR in rgb to capture the source by scene capture component
  + Used Lumen Global Illumination for all lights
  + Changing render target format (tried RTF RGB A8 SRGB)

* Result : Inspection object started rendering, but only the holes were visible.
* Experimented with changing the light source, discarded the torus emissive light and put on directional light.
* Result : object started to render completely. Conclusion : there must be some issue with the light source, look into it.
* Still, there are a lot of issues with the renders, such as:
  + Light source issue
  + It is not giving the expected metallic essence and illuminating the expected parts of the object.
  + The image rendered on camera is way different (illumination difference) that the one rendered in editor
* Conclusion : There must be some issue with the shader, recheck it. Discard programmable Custom nodes and use simplify nodes. Then, get into the lighting issue. Take and save the rendered target textures (export them in photo format with desired resolution), and also take the screenshots using camera simulation screenshot method given by unreal to compare.