CEng 491 – Project KickOff Document

Abdullah Mert Tunçay
Barış Suğur
Zumrud Shukurlu
Batuhan Bat

7 November, 2018

“GRT” KickOff Document

1 Description

The number of modern “mixed” renderers that can utilize both rasterization and ray tracing to get the best performance/quality ratio is very low. NVIDIA’s latest steps in the last few years to support the DXR and specifically developed hardware is a clear evidence that real time renderers will involve more and more ray tracing in the future.

Most renderers that is available to end-users are either offline renderers (can not be used in real time) or real-time renderers that doesn’t use ray tracing technique at all.

In our project, we will create a GPU based real time Ray Tracing API that aims to solve the famous Global Illumination problem. The end product will use a masking algorithm to separate the parts of the scene where ray tracing will be used and render small to medium-sized scenes in real time with high tier GPUs. The target audience of the product is Computer Graphics developers.

2 Master Feature List

1: High Performance
The API will be implemented with CUDA to work on GPU so GPU’s computational power will be used which will make the end product run much faster compared to a CPU based one.
2: Improved Quality of Lighting
With the added ray tracing technique, the quality of lighting in the render will improve vastly compared to a direct lighting one.
3: Dynamic Scene Support
Dynamic scenes support will be implemented alongside with BVH structure.
4: Rendering Real Time Dynamic Scenes
Dynamic Scenes will rendered fast enough to reach near real time experience. The run time performance will be between 20 – 30 fps)
5: Acceleration Structure Technique (BVH) Algorithm
A chosen suitable acceleration structure technique will be implemented to reduce the computation time.
6: Area and Punctual Light Support
Different type of light sources (area light and punctual light) will be supported in the scenes.

7: Solution for Global Illumination Problem
Using the Monte Carlo path tracing technique the global illumination problem will be solved.
8: Optimal Quality/Performance Ratio
End product will be a hybrid renderer which can utilize both rasterization and ray tracing to get the best performance and quality.
9: Increased Performance With Sampling
With implementation of the sampling mechanism, the final quality of the image will increase while keeping the number of samples constant.
10: NVIDIA GPU Support
Because CUDA will be use for GPU implementation, the API will support NVIDIA’s GPUs.
11: Real Time Ray Tracing Renders for Small-Medium Sized Scenes
The end product will render small to medium sized scenes with real time performance (20-30 fps) on high tier GPU’s.

3 Workpackages

In this section, work packages of the project are listed.

WP	Term	WP title	Number of person-months
1	491	Project planning and architecture design	3
2	491	Proof of concept implementation in CPU	3
3	491	Transferring the implementation to GPU	3
4	491 & 492	Acceleration structure (BVH) implementation on GPU	8
5	492	Monte Carlo Global Illumination	8
6	492	Implementing a sampling mechanism	3
7	492	Testing the performance with offline renderers	2
		Total:	30

Detailed Descriptions of High-Level Workpackages

3.1 WP1 - Project planning and architecture design

In this workpackage, the following functionalities / features / work items will be implemented;

1. Develop the list of master features of the project.
2. Produce project development plan in accordance with Master Feature List.
3. Design the overall architecture of the project.
4. Analyze risks and make a management plan.

5. Prepare the Kick-off Document of the project.

3.2 WP2 – Proof of concept implementation in CPU

In this workpackage, the following functionalities / features / work items will be implemented;

1. Implement a mask algorithm to choose the objects that will be ray traced and apply rasterization to the rest of the scene.
2. Design a general purpose ray tracing API that uses the mask algorithm in C#/C++ on Unity.
3. Test the initial implementation of Ray Tracing API using some of the scenes provided by Unity.

3.3 WP3 - Transferring the implementation to GPU

In this workpackage, the following functionalities / features / work items will be implemented;

1. Research of CUDA (Compute Unified Device Architecture) to understand GPU programming basics and analyzing example GPU based renderers on Unity.
2. Implementation of the ray tracer on GPU.
3. Comparing the performances of CPU and GPU implementations.

3.4 WP4 - BVH implementation on GPU

In this workpackage, the following functionalities / features / work items will be implemented;

1. Experimenting and finding the optimal acceleration structure algorithm to reduce the computation time.
2. Integrating the acceleration structure algorithm to our product.
3. Implementation of dynamic scene support.
4. With the BVH structure, dynamic scenes will be rendered in real time.

3.5 WP5 - Monte Carlo Global Illumination

In this workpackage, the following functionalities / features / work items will be implemented;

1. Research on Direct, Indirect Lighting and Light Source Type Monte Carlo Implementations.
2. Implementing Direct and Indirect lighting by Path Tracing methods.
3. Implementing the support of Light Source Type (object as light sources).
4. Testing and updating the API.

3.6 WP6 - Implementing a sampling mechanism

In this workpackage, the following functionalities / features / work items will be implemented;

1. Research on sampling mechanisms methods such as Light Sampling and Cosine Sampling.
2. Choosing and implementation of the most suitable sampling method.

3.7 WP7 - Testing the performance with offline renderers

In this workpackage, the following functionalities / features / work items will be implemented;

1. Searching an finding optimal offline renderers (such as Mitsuba etc.) to compare with our API.
2. Comparing the performances of our product and the chosen offline renderer.

4 Bonus Workpackages

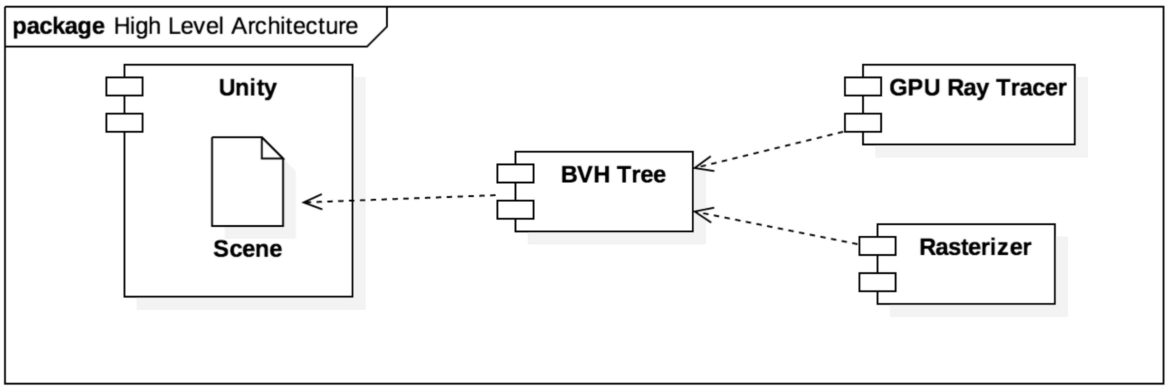
Workpackages given below will be implemented if all mandatory workpackages done before the schedule.

1. **Fur Rendering:** Implementation the support of hair like (human hair, animal fur etc.) objects.

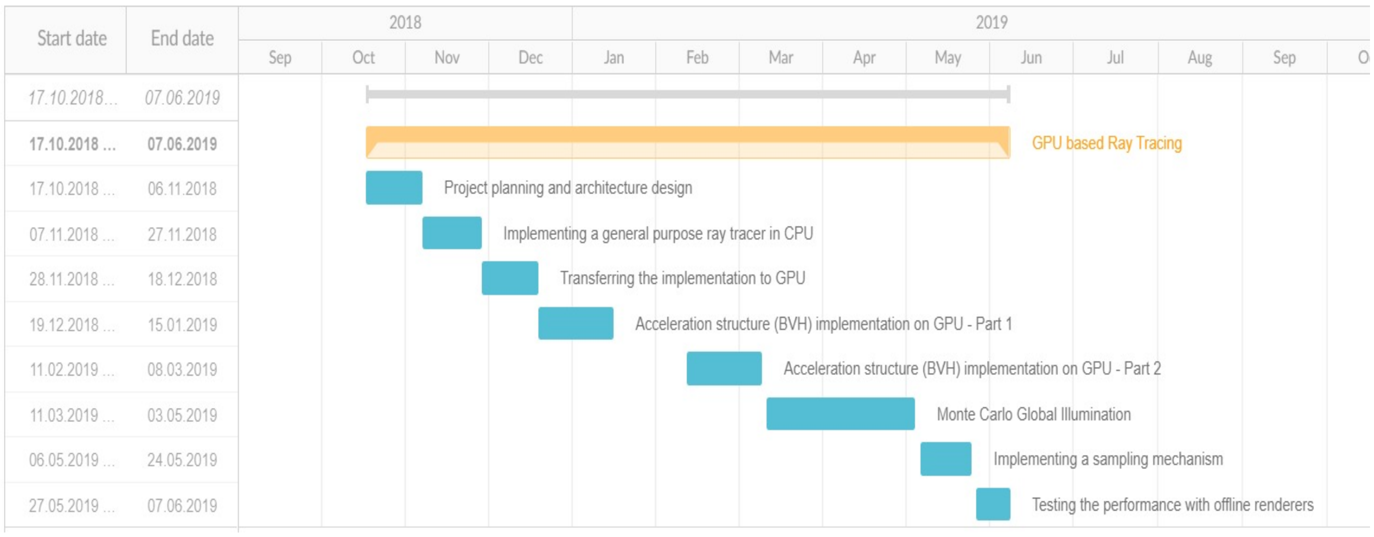
2. **Implementation of Recent BVH Techniques:** Analyzing the comparison between different recent BVH techniques and implementing a different BVH.

3. **Noise Reduction:** Reducing noise in the final image using Image Processing techniques.

5 Overall Systems Architecture



TimeLine



Start date	End date	
17.10.2018...	07.06.2019	
17.10.2018 ...	**07.06.2019**	GPU based Ray Tracing
17.10.2018 ...	06.11.2018	Project planning and architecture design
07.11.2018 ...	27.11.2018	Implementing a general purpose ray tracer in CPU
28.11.2018 ...	18.12.2018	Transferring the implementation to GPU
19.12.2018 ...	15.01.2019	Acceleration structure (BVH) implementation on GPU - Part 1
11.02.2019 ...	08.03.2019	Acceleration structure (BVH) implementation on GPU - Part 2
11.03.2019 ...	03.05.2019	Monte Carlo Global Illumination
06.05.2019 ...	24.05.2019	Implementing a sampling mechanism
27.05.2019 ...	07.06.2019	Testing the performance with offline renderers

Risk Assessment

Risk #	Description	Possible Solution(s)
1	Lack of debugging tools	Porting the code to CPU and testing it there.
2	Not reaching the real time performance in complicated dynamic scenes	Searching for better BVH or sampling methods and trying the execution on higher tier GPU’s.