



National Taiwan University

Parallel Programming, Fall 2025

Physics Based Scene Rendering

Final Project Proposal

Group n°20 :

BALOT Louise (白芷琳), T14902202
GUILLOUX Kévin (高凱瑞), A14943201
LY Gregory (李金發), T14902203

Instructor :
Chun-Yi Lee

Teacher Assistant :
Yi-Chen Wu

Summary

I Motivation 1

II Target Topics 1

III Time and organization schedules 2

 III.1 Task scheduling 2

 III.2 Task repartition 2

IV Scope and limitations 3

List of Figures

Figure 1 AI representation of a possible rendered frame 1

Figure 2 Schedule of the program behavior 2



I Motivation

The base of every 3D engine, being for video games or for movies animation is to output an image based on physical interactions between a world and some objects that are living inside of it.

As such engines are often really complex and can work like “black boxes”, we are in this project aiming at reproducing basic aspects of such engines, optimized entirely in CUDA to take advantage of GPUs and multicore CPUs.

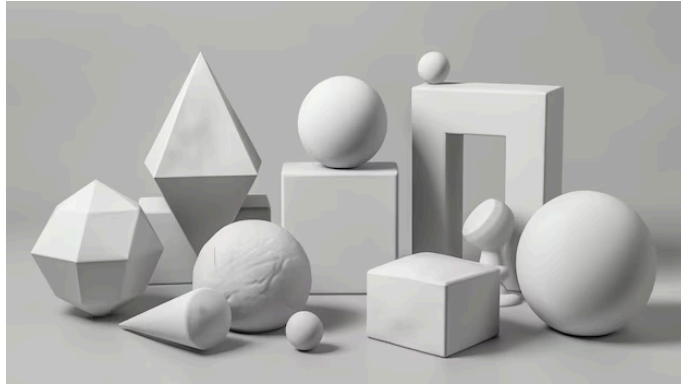


Figure 1 : AI representation of a possible rendered frame

This kind of engine has the advantage of being iterative, with multiple parameters that can be activated or not to modify both the quality and precision of the output, and the time needed to render each frame. Giving a good choice of tradeoff between low quality real time rendering and realistic but computationally intensive rendering.

II Target Topics

As this project aims at developing a working physics based rendering engine, it will include many topics of computing :

- **Physics approximation** : In order to keep the computation time in check, we will need to use approximate physics interactions between objects. If time is on our side, we will also include ways to improve or decrease these calculations precisions in order to emphasize their impact on performance.
- **Real time rendering** : As we will try to aim at creating a real 3D engine, we need it to be able to compute the full length of an action, and output a view of what is happening in real time, not after the end of the computation.
- **CPU/GPU Load balancing** : Using CPUs to compute physics and GPUs to render frame is not something new, but it is still important to produce a good load balancing in order to improve the performance of the renderer compared to a fully sequential implementation.
- **Scene Rendering optimizations** : As we want our scene renderer to be able to output multiple frames per second, we need to use clever techniques to compute the pixels in the image efficiently. This will be at the prize of precision, but, if time allows it, we will allow for ways to improve or decrease the rendering quality.

III Time and organization schedules

III.1 Task scheduling

The parallel computation of this task will use both the CPU and GPU, and will aim at using all of the available resources on both of them. The following schedule shows the behavior of the main computation loop of this program :

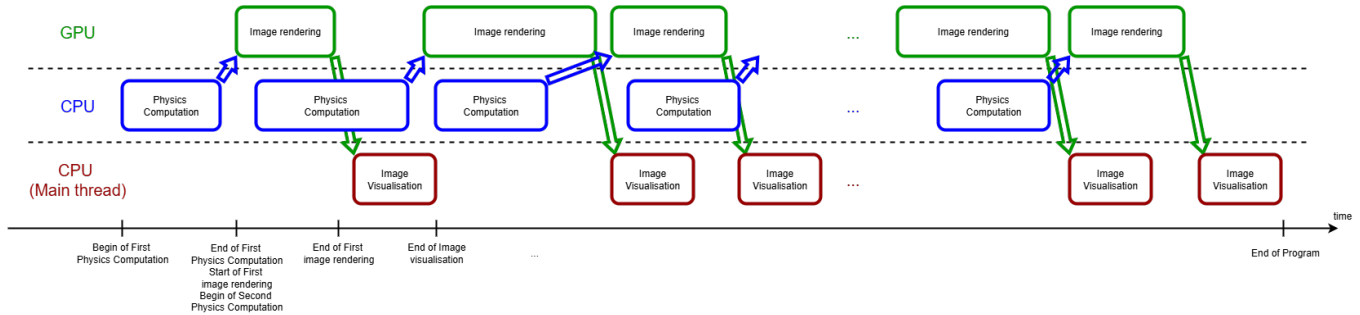


Figure 2 : Schedule of the program behavior

We can see that the scheduling gets into a kind of pipelining, showing the interest of such a scheduling for a process that needs to be repeated a really big number of time, in order to create either a video, or to be outputed in real time through multiple images.

III.2 Task repartition

As we are a group of 3 students, we will share the workload by types of computation in the program, as they are relatively independant from one another, except from their input and output datas.

- **CPU Physics computation** : One student will incorporate all of the necessary functions used for the physics computations that need to happen in order to calculate the new position of the objects.
- **GPU image renderer** : One student will use CUDA in order to offload the image rendering to the GPU (or GPUs), according to a set of positions given by the CPU.
- **Real time image visualisation** : One student will get to work on the main skeleton of the code (gathering the work of both the previous students) and package the image into a suitable real time visualisation window, with all kind of metrics to track performance.
- **Sequential version** : In order to have comparison material against a sequential version of our engine, the student with the least amount of work will convert parallel codes into sequential ones and run the benchmarks.

Additionally to the previous points, each student will produce different versions using GitHub in order to improve one of the following aspects of their code :

- **Precision** : The ability to be the closest to a real life representation of the scenery.
- **Performance** : The ability to output a new frame in the least amount of time (includes both physics computation and frame rendering).
- **Configurability** : The ability to allow the user to modify predefined variables in order to get a different tradeoff between precision and performance.

IV Scope and limitations

In order to keep the development possible and be able to produce something that gives a satisfactory result, as well as showing a persistent speedup when compared to it's sequential counterpart, we are going to set some boudaries :

- **Fixed number of objects** : As we want to keep the scheduling of physics computation and the memory management relatively simple in this project, the number of objects to be simulated in the space will either be choosen at compilation or at launch time. It won't be possible to add objects while the program is running.
- **Fixed camera position and view** : Because we only want to show what happens into a fixed space, we won't allow for camera movements, as it helps us focus on other aspects of scene rendering.
- **Compilation fixed parameters** : At first, all of the physics and rendering parameters will be fixed at compilation to ease memory management and functions initialization. Later, we may allow for execution time parameter changes if time allows it.
- **Simple geometric shapes** : As we won't use polygons for the rendering of the 3D image, we will start with very simple geometric shapes for the objects in order to keep both CPU and GPU computation relatively easy (cubes, spheres, ...).

