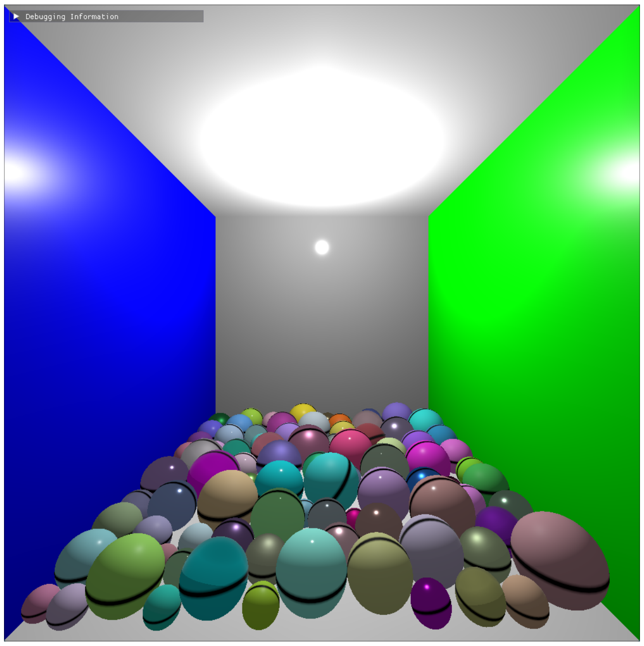
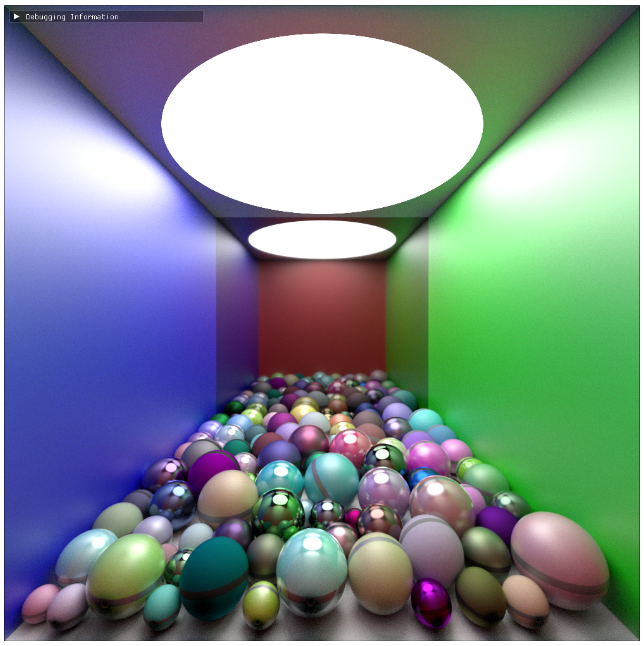
ÉCOLE POLYTECHNIQUE DE LOUVAIN

Computational Geometry – **LMECA2170**
Université catholique de Louvain
2024-2025

Homework 2: Spatial Data Structures





The goal of this second assignment is to implement spatial data structures to effectively improve the performance of geometric operations within numerical software.

You are given a working implementation of software simulating spheres moving around in a box, affected by gravity and collisions between each other. Your goal is to accelerate the code used to find these collisions.

The deadline for the assignment is December 7th, 2025. You may submit your code by email at kilian.verhetsel@uclouvain.be. You may also use this email to ask further questions about the homework.

General instructions:

1. Form groups of three students. If not possible, groups of two are allowed. Groups of four are not permitted (please do not ask for exceptions).

2. You are allowed to use AI tools at any time. However, the evaluation of your project will be based not only on the performance of your implementation, but also on an interview of each group with all instructors. It is therefore absolutely essential that you both verify what the AI provides and understand it thoroughly, down to the smallest details.

3. The code will be graded on 20 points according to the structure detailed in this document. This grade is initially attributed to the group. The interview will allow us to objectify this grade and to adjust it individually, depending on each member's performance during the discussion. Adjustments may be made both downwards (in case of insufficient understanding) and upwards (in case of outstanding individual insight).

4. You must submit a short report of at most one page explaining how you organized your work. Be precise about "who did what". Each member of the group must sign this report.

5. **Important notice.** We will use all anti-plagiarism tools at our disposal. Any confirmed case of plagiarism will result in a grade of zero for both the group who copied and the group whose work was copied.

Program Structure

Your program consists mainly of one file, `homework.py`, where you must replace the `find_intersections` method with an efficient implementation. To run the program, you first need to install the dependencies, for example using pip: `pip install -r requirements.txt`. The provided implementation checks every pair of objects, and is therefore very slow when the number of objects gets large. You may use any spatial data structure of your choice to achieve better performance.

You may find that you need to change your data structures after the insertion of new objects, or at the start of the simulation. You may do so by redefining the `on_size_changed` and `__init__` methods, respectively.

You are given the current state of the simulation using a set of NumPy arrays:

- `radii`, the sizes of the N spheres in the simulations;
- `positions`, of shape $(N, 3)$, the positions of these spheres;
- `rotations`, of shape $(N, 4)$, the orientations of these spheres encoded as a quaternion [1];
- `velocities`, of shape $(N, 3)$;
- `angular_velocities`, of shape $(N, 3)$.

The method computing the intersection between a pair of objects is provided to you (`intersect(a, b)`). Your implementation should return a list populated with the return values of this method, with one element per contact point that you identified.

In your implementation, you are free to use parallel computing or to offload computations to a C implementation. You must, however, implement the data structure yourself (e.g. you cannot use a KDTree implementation from an existing library). A correct implementation of a spatial data structure, with sub-quadratic complexity (i.e. it is faster than the naive version for *some* amount of points) covers 10/20 points of your final grade. 5 additional points concern the practical efficiency of your implementation (i.e. your implementation is faster for practical sizes, it enables larger real-time simulations, etc.).

Extension: GPU Traversal (+5 points)

The graphical interface used to display the simulation allows you to enable a path tracer to display the simulation. The provided implementation is also inefficient: it computes the intersection of light rays with every single sphere in the simulation. For the second part of the homework, you may modify the `shaders/homework.glsl` file to use a spatial data structure that you implemented. You are not required to build the data structure on the GPU directly. It is possible to transfer the data structure you used for the first part (example code is provided), and you only need to write a function to query it on the GPU.

Specifically, on the GPU, you need to reimplement the `trace_ray` function which, given a ray $\mathbf{o} + t\mathbf{r}$, must find the closest sphere that intersects with the ray. The code you were provided with includes a function to compute the intersection between a ray and a single sphere (`ray_ball_intersection`) or the planes of the box (`ray_plane_intersection`). In your Python program, you need to use the WebGPU API [2] to transfer or build a spatial data structure in the `update_gpu_data` method. The code provided to you already shows how to copy an array of random numbers.

[2]https://en.wikipedia.org/wiki/Quaternion
[2]https://wgpu-py.readthedocs.io/en/stable/