Week 8 Challenge: MPCS 51042-1 2018

Due: Never

Introduction

The concept of ray tracing is explained in the included file, `ray_tracing_theory.pdf`, an old homework I wrote when I was TAing High Performance Computing. The theory and implementation are still directly applicable to this task.

Take a look at the "Implementation" section of `ray_tracing_theory.pdf` and compare it to the implementation in `ray_tracer.py`. The two should be quite close to each other.

Even though `ray_tracer.py` makes use of NumPy vector operations, it does not necessarily do it efficiently. This is because vector operations such as dot product and `abs` are performed on very short 3D vectors. **An ideally optimal implementation would vectorize the entire for loop. Your task is to attempt this.**

Approach

Consider first generating a large array of random rays and then performing vectorized operations on that array. For this approach, there are two likely layouts for the array of rays. These are both common design patterns in vector programming:

"Array-of-structs": In this approach, each vector (x, y, z) is grouped in dimension 1; and the n vectors are in dimension 0. For example:

```
[[x0, y0, z0], [x1, y1, z1], ..., [xn, yn, zn]]
```

"Struct-of-arrays": In this approach, all x components are grouped in dimension 1; likewise for y and z components. Then these 3 arrays are in dimension 0. For example:

```
[[x0, x1, ..., xn], [y0, y1, ..., yn], [z0, z1, ..., zn]]
```

The struct-of-arrays layout might seem counter-intuitive. However, it can allow many vector operations that are difficult or impossible to do with the array-of-structs approach. Try experimenting with struct-of-arrays to see what’s possible.