

# Raytracer Optimisation

Aaron Kallio

December 17, 2025

## 1 Introduction

For this assignment I have made the raytracer run for just 1 frame and take a screenshot using the stbi library found on GitHub. The user can enter the width, height, number of rays per pixel and number of spheres. The amount of bounces I have hard-coded as 5 and the camera cannot move.

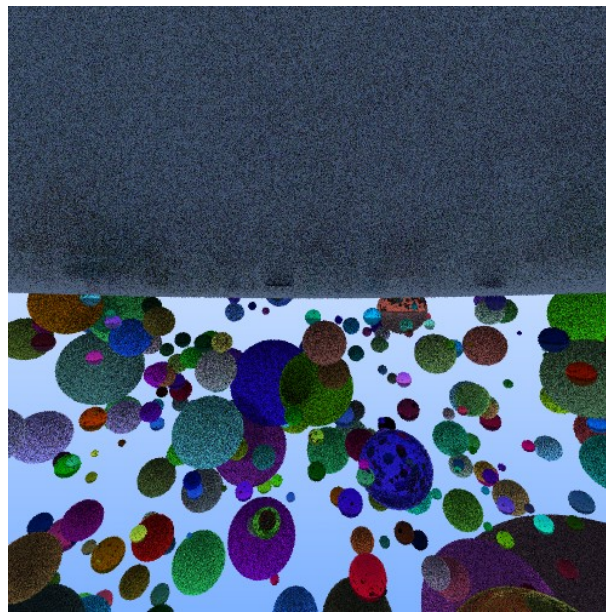


Figure 1: Example of image generated by the raytracer. The images are created upside down but as it was for testing, I focused on optimising the raytracing instead of flipping the image.

## 2 Raytracing Optimisation

### 2.1 Before Optimising

Before any optimisation I had to tweak the cMake file slightly and after I got the code to working I ran Visual Studios static code analysis and 2 different profilers (Visual Studios built in one and the Intel Profiler).

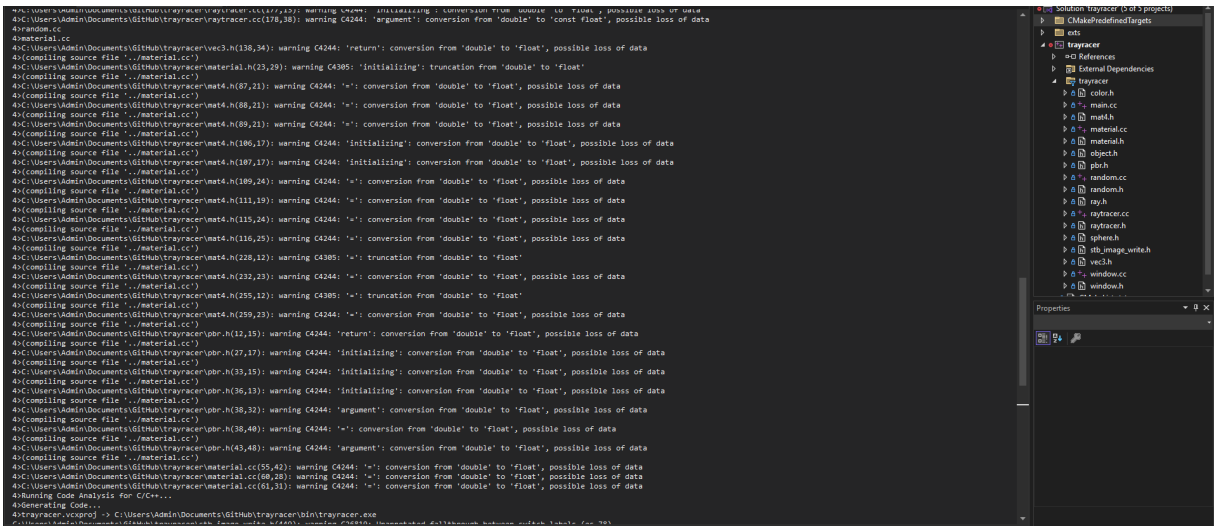


Figure 2: Static analysis.

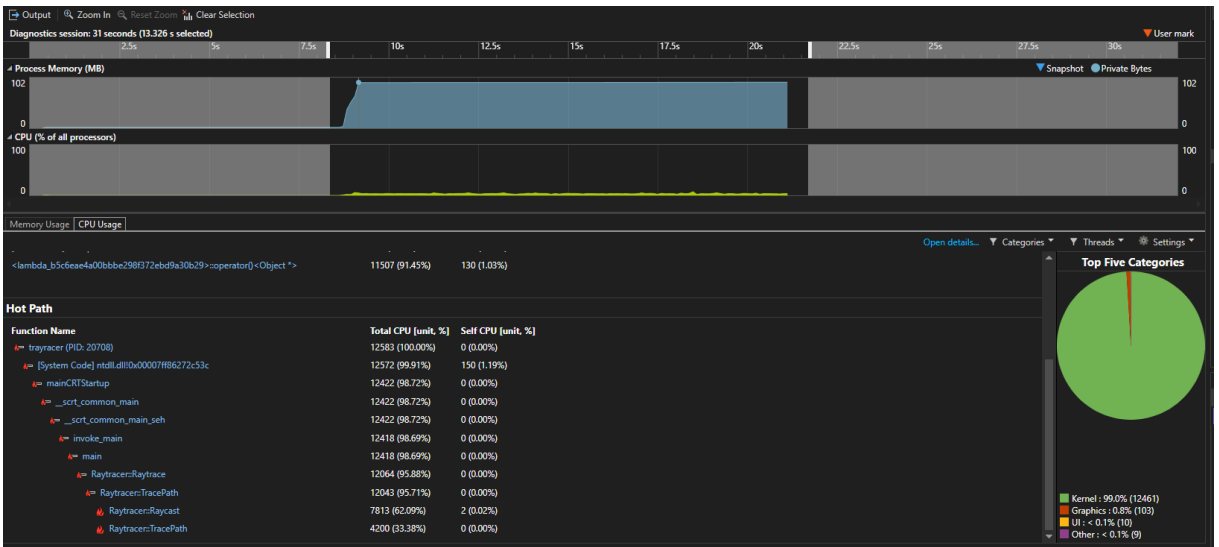


Figure 3: Visual Studio profiler.

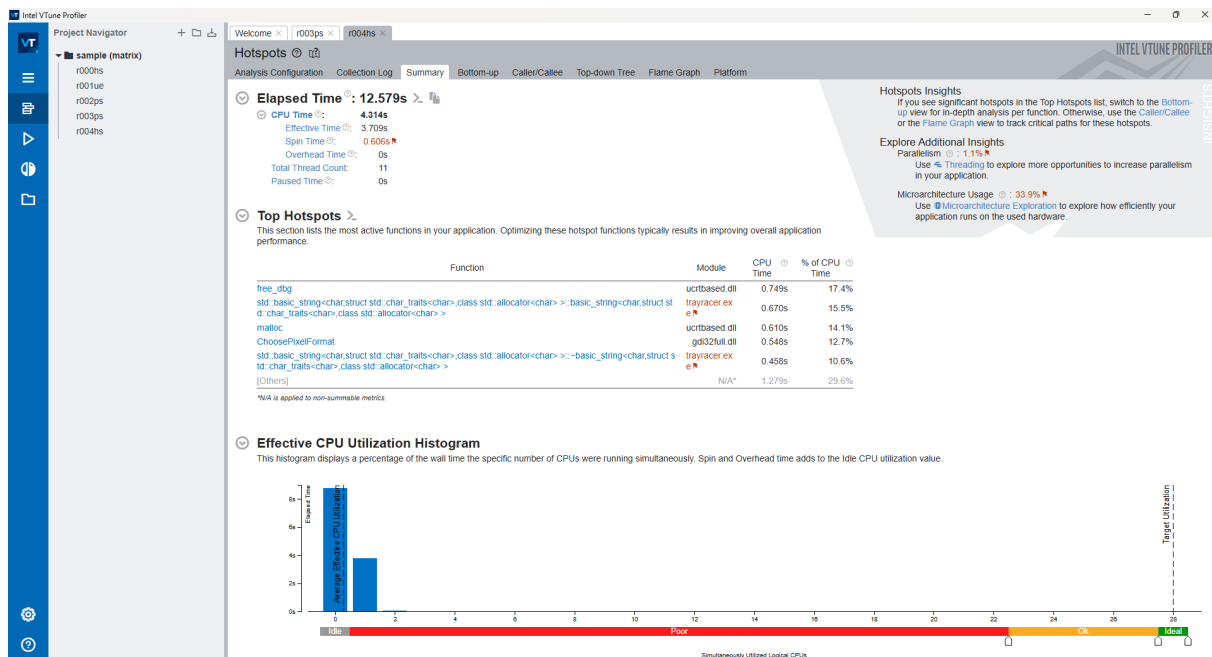


Figure 4: Intel profiler.

## 2.2 Optimising

The first optimisation was to remove unnecessary code that didn't affect the result in any way. The biggest one that I found was a sort that was performed on the object array that was ran for every pixel. I have not noticed any changes in the look of the image from removing such unnecessary code.

```
Enter Width: 100
Enter Height: 100
Enter Rays Per Pixel: 1
Enter Sphere Amount: 10
Elapsed time in seconds is 0.343446
Ray amount is 15249
Rays per second is 44400

C:\Users\Admin\Documents\GitHub\trayracer\bin\trayracer.exe (process 2964) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 5: After removing the necessary code such as the sort.

The next optimisation I did was changing the type in material.h from a string to an int. For a more easy to use data type I should have used an enum but as I didn't have to use it after this optimisation, I saw no reason to change it. Another similar change I did was to make the program use the Object classes int id instead of char\* name as it will make assigning and making unique objects more performative. Doing almost anything with an int is faster than strings.

```

Enter Width: 100
Enter Height: 100
Enter Rays Per Pixel: 1
Enter Sphere Amount: 10
Elapsed time in seconds is 0.357893
Ray amount is 15249
Rays per second is 42607.7

C:\Users\Admin\Documents\GitHub\trayracer\bin\trayracer.exe (process 13056) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|

```

Figure 6: After removing the string handling.

For every pixel, the object array was rebuilt which is unnecessary so I made it so that at the start the array is built once and is iterated through. This would still work for multiple frames aswell. Having to rebuild and remove from an array will cause slowdown as the memory will be resized and moved as it is a c++ vector meaning dynamically allocated. I'm still using vector but from my research, with my current usage it should be as fast as a static array but easier to use.

```

Microsoft Visual Studio Debug
Enter Width: 100
Enter Height: 100
Enter Rays Per Pixel: 1
Enter Sphere Amount: 10
Elapsed time in seconds is 0.184771
Ray amount is 15249
Rays per second is 82529.1

C:\Users\Admin\Documents\GitHub\trayracer\bin\trayracer.exe (process 26680) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

Figure 7: Array made just once.

One way I improved the maths in the program was lowering the calculations done in sphere.h intersect() function. This didn't hugely increase performance but it still had a noticeable impact as less if statements are also present. I do now have to normalise ray.m but it is not too taxing.

The biggest change was using multithreading by processing each row in batches which speed things up as more cores are used. This however does slow down the very small aspect ratios such as 100x100 but for this sort of image creation you are more likely to make a large image anyways.

```
Optional<HitResult> Intersect(Ray ray, float maxDist) override
{
    constexpr float minDist = 0.001f;
    float len = sqrt(
        ray.m.x * ray.m.x +
        ray.m.y * ray.m.y +
        ray.m.z * ray.m.z);

    vec3 dir = {ray.m.x / len, ray.m.y / len, ray.m.z / len}; //normalise ray.m for reduced calculation
    vec3 oc = ray.b - center;
    float b = dot(oc, dir);
    float c = dot(oc, oc) - radius * radius;
    float discriminant = b * b - c;

    // No intersection if discriminant < 0
    if (discriminant < 0.0f)
        return Optional<HitResult>();

    float sqrtDisc = sqrt(discriminant);
    float temp = -b - sqrtDisc;
    float temp2 = -b + sqrtDisc;
    float t = (temp > minDist && temp < maxDist) ? temp : temp2; //if temp is applicable then use temp otherwise use temp2
    if (!(t > minDist && t < maxDist)) return Optional<HitResult>();

    // Compute hit
    HitResult hit;
    hit.t = t;
    hit.p = ray.b + dir * t;
    hit.normal = (hit.p - center) * (1.0f / this->radius);
    hit.object = this;

    return Optional<HitResult>(hit);
}
```

Figure 8: intersect code.

```
Enter Width: 500
Enter Height: 500
Enter Rays Per Pixel: 5
Enter Sphere Amount: 5000
Elapsed time in seconds is 117.699
Ray amount is 7332682
Rays per second is 62300.5

C:\Users\aaaron\OneDrive\Documents\GitHub\trayracer\build\trayracer.exe (process 64600) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 9: Multithreading test on my far slower laptop.

### 3 Last Thoughts

I have increases the performance by atleast 100x for the large sized images but it still does somewhat exponentially increase in time with size. There is still some room for improvement such as using SIMD but I could not find a way to implement it without breaking everything.

### References