TRACE_IT: Assignment 2

Interactive Ray Tracer with Multiple Spheres

June 2, 2025

1 Assignment Overview

In this assignment, you will extend the basic ray tracer implementation from *Ray Tracing in One Weekend* (Sections 1-6) to create an interactive command-line application. Your program will allow users to specify the number of spheres, their properties (radius and color), and render the scene with surface normal-based shading.

Basically you are free to copy paste the code from the book(Ray tracing in a weekend) or can try writing yourself after understanding it. The main purpose is, use you should be able to understand how things are setup in this virtual world of ours. Try seeing how each class interacts with the other, what each class, object does and how they come together to create images from just simple lines of codes.

This command line based program will later be extended to include more complex behaviour in later assignments.

2 Technical Requirements

2.1 Core Functionality

Your ray tracer must implement the following features based on the book content:

1. **Ray-Sphere Intersection**: Implement the mathematical intersection test between rays and spheres as described in the book
2. **Surface Normal Calculation**: Compute outward-facing normals at intersection points
3. **Multiple Object Handling**: Use a "hittable list" to manage multiple spheres and find the closest intersection
4. **Normal-based Shading**: Map surface normals to RGB colors for visualization
5. **PPM Image Output**: Generate images in the Portable Pixmap format

2.2 Command Line Interface

Your program should accept the following command-line arguments, you are free to modify them.

```
./raytracer [OPTIONS]

Options:
  --spheres N          Number of spheres to render (1-10)
  --radius R1,R2,...   Comma-separated list of sphere radii
  --color C1,C2,...    Comma-separated list of sphere colors
  --output FILENAME     Output PPM filename (default: output.ppm)
  --width W             Image width in pixels (default: 400)
  --height H            Image height in pixels (default: 400)
  --help                Show usage information
```

2.3 Color Specification

Support the following color formats:

- **Named colors**: red, green, blue, yellow, cyan, magenta, white, black
- **RGB values**: Format as “r,g,b” where each component is 0-255
- **Random**: Use keyword “random” to generate random colors

2.4 Required Classes/Structures

Implement the following key components:

```cpp
struct Vec3 {
    float x, y, z;
    // Vector operations: +, -, *, dot, cross, normalize
};

struct Ray {
    Vec3 origin;
    Vec3 direction;
    Vec3 at(float t) const;
};

struct HitRecord {
    Vec3 point;
    Vec3 normal;
    float t;
    bool hit;
};

class Sphere {
    Vec3 center;
    float radius;
    Vec3 color;
public:
    bool hit(const Ray& ray, float t_min, float t_max,
             HitRecord& record) const;
};

class Scene {
    std::vector<Sphere> spheres;
public:
    bool hit(const Ray& ray, float t_min, float t_max,
             HitRecord& record) const;
};
```

Listing 1: Essential Data Structures

3 Program Specifications

3.1 Default Scene Setup

- Camera positioned at origin (0, 0, 0)
- Looking towards negative z-axis
- Viewport: 4 units wide, 4 units tall, 1 unit away from camera
- Background: Linear gradient from blue to white (as in the book)

3.2 Sphere Positioning

When multiple spheres are specified:

- Distribute spheres randomly within the viewing volume
- Ensure spheres don't overlap (maintain minimum distance between centers)
- Place spheres at varying depths (z-coordinates from -5 to -15)

3.3 Input Validation

Your program must handle: If you are finding it difficult to implement. Assume the inputs are already validated and correct.

- Invalid command-line arguments
- Mismatched number of spheres, radii, and colors
- Invalid file paths

4 Sample Usage Examples

```
# Single red sphere
./raytracer --spheres 1 --radius 1.0 --color red

# Three spheres with different properties
./raytracer --spheres 3 --radius 0.5,1.0,1.5 \
           --color red,green,blue --output scene1.ppm

# Random scene
./raytracer --spheres 10 --radius random --color random \
           --width 800 --height 600 --output random_scene.ppm

# High-resolution single sphere
./raytracer --spheres 1 --radius 2.0 --color 255,128,64 \
           --width 1024 --height 1024 --output hires.ppm
```

Listing 2: Example Command Line Usage

5 Deliverables

5.1 Code Submission

1. Complete C++ source code a readme file containing demo commands to test.

6 Resources and References

- Shirley, Peter. *Ray Tracing in One Weekend*, Sections 1-6

7 Submission Guidelines

- Submit via google form as a single ZIP file
- Google form link : https://forms.gle/T2wtrn3nzfm7BMA2A