Preparatory Lab: Minimal Ray Tracer in Modern C++

Overview

This lab is a preparatory project for an advanced computer graphics course. The goal is to build a minimal ray tracer in modern C++ in order to become comfortable with:

- Modern C++ programming idioms (RAII, classes, const correctness)
- Core rendering mathematics
- Ray–object intersection
- Recursive ray tracing and materials

The focus is on clarity, correctness, and understanding rather than performance or advanced features.

Time Budget

- Duration: 3 weeks
- Workload: 1–2 hours per day
- Total effort: approximately 25–35 hours

Technical Constraints

- Language: C++17 or newer
- Build system: CMake
- No global state
- No raw `new` / `delete`

Allowed libraries:

- `glm` (optional, for math)
- `stb_image_write.h` (for PNG output)

No graphics APIs (OpenGL, Vulkan, DirectX) may be used.

Final Target Features

Core Infrastructure

- `Vec3` class with arithmetic operators
- `Ray` structure
- `Image` class managing pixel storage via RAII
- PNG image output

Camera

- Perspective camera
- Configurable field of view and aspect ratio

Geometry

- Sphere primitive
- Infinite plane primitive

Materials

- Lambertian (diffuse) material
- Perfect mirror (specular reflection)

Rendering

- Recursive ray tracing with a fixed depth limit
- Basic lighting
- Gamma correction

Recommended Class Structure

- `Vec3`
- `Ray`
- `Camera`
- `Image`
- `Hittable` (abstract base class)
- `Sphere`, `Plane`
- `Material` (abstract base class)
- `Lambertian`, `Mirror`

Daily Schedule and Checklists

Each checklist corresponds to approximately 1–2 hours of focused work.

Week 1: C++ Foundations and First Image

Day 1

□ Create CMake project

□ Verify clean build

□ Implement `Vec3` class

□ Implement `Ray` structure

Day 2

□ Implement `Image` class with RAII

□ Store pixels in `std::vector`

□ Write PNG output function

□ Generate a test image

Day 3

□ Implement ray–sphere intersection

□ Define hit record structure

□ Render a single sphere

Day 4

□ Implement perspective camera

□ Add configurable FOV and aspect ratio

□ Render multiple spheres

Day 5

□ Remove global variables

□ Add `const` correctness

□ Clean class interfaces

Milestone: Render a stable image of multiple spheres from an arbitrary camera position.

Week 2: Materials and Recursion

Day 6

□ Create abstract `Material` base class

□ Implement Lambertian material

□ Verify correct shading

Day 7

□ Implement recursive ray tracing

□ Add depth limit

□ Add shadow rays

Day 8

□ Implement perfect mirror material

□ Debug reflection directions

□ Verify numerical stability

Day 9

□ Implement infinite plane primitive

□ Add plane to scene

□ Verify intersections

Day 10

□ Implement gamma correction

□ Clean recursion logic

□ Improve image stability

Milestone: Diffuse and reflective objects render correctly.

Week 3: Polish and Understanding

Day 11

□ Implement simple anti-aliasing

□ Add jittered sampling

Day 12

□ Refactor code for readability

□ Improve naming and layout

Day 13

□ Add simple scene description

□ Load objects from text or structured input

Day 14

□ Build a Cornell-style test scene

□ Verify lighting consistency

Day 15

- □ Final cleanup
- □ Remove dead code
- □ Comment design decisions

Final Milestone: A clean, minimal ray tracer that is fully understood end-to-end.

Notes

If a task exceeds two days, it should be removed from scope. Depth of understanding is prioritized over feature completeness.