CPSC 4050/6050 Fall 2021

Project 2

due date: October 1 upload to handin

Summary

For this project you will write, compile, and execute C++ code to raytrace a simple 3D scene and write the image to a simple ascii image file. The output image will be in ppm format. You will put all of the code and the output image in a folder, zip it into a single zip file, and upload it to . Your submission will be compiled and executed on School of Computing linux computers. Compilation will be accomplished via `make`. It is very much in your interest to make sure your code compiles and runs in that computing environment.

1 Description for All Students

As we discussed in class, ray tracing a simple scene requires the following elements:

- A camera
- An Image Plane (a collection of pixels in a rectangular arrangement)
- A Ray (for each pixel of the image plane)
- One or more 3D objects (planes, spheres, triangles), with material properties (e.g. color)
- Zero or more lights associated with each 3D object.
- A Scene, which is a container of 3D objects.
- A raytracing function

To create your ray tracer, you must create a separate C++ class for each of the following concepts:

Class	**Class Data** (minimum required)	**Class Methods** (minimum required)
Camera	• position • view direction • up direction • field-of-view • aspect ratio	`Vector view(float x, float y) const;`
ImagePlane	• Nx, Ny • Color* data	• `Color get(int i, int j) const;` • `void set(int i, int j, const Color& C );`
Ray	• position • direction	• `const Vector& get_position() const;` • `const Vector get_direction() const;`
Sphere	• position • radius • color	• `float intersection(const Ray& r) const;` • `const Color get_color() const;` • `Color shade(const Vector& P, const Light& l) const;`
Plane	• position • normal direction • color	• `float intersection(const Ray& r) const;` • `const Color get_color() const;` • `Color shade(const Vector& P, const Light& l) const;`
Triangle	• vertex0 • vertex1 • vertex2 • color	• `float intersection(const Ray& r) const;` • `const Color get_color() const;` • `Color shade(const Vector& P, const Light& l) const;`
Light	• position • color	• `const Vector& get_position() const;` • `const Color& get_color() const;`

The ray tracing function is not a class:

```
Color Trace( const Ray& r, const Scene& s );
```

This function tests the intersections of the input Ray with each object in the Scene container. If any of the objects are intersected by the ray, the Trace

function returns the color that is computed by the `shade(P,l)` method of the *closest* object. The `shade` method takes as input the location of the intersection and a light object. The particular style of shading needed for this assignment is Lambertian reflection. If no objects are intersected, `Trace` returns black.

A ray-trace renderer performs the following 4 steps for each pixel of the image plane:

1. Call the camera's `view(x,y)` method with the `x,y` for the given pixel, returning the pixel direction vector.
2. Initialize a ray with the position of the camera and the pixel direction.
3. Call the `Trace` function, which returns a color following the outcome of its intersection tests (as described above).
4. Set the color of the pixel to be the color returned by `Trace`.

You must write C++ code for each of the classes in the above table. The classes may have more data and/or methods beyond what is listed in the table, but those data and methods must be in the declaration and implementation of each of the classes. You must create separate header and implementation files for each class, i.e. you will have at least 8 header files and at least 8 implementation files, with names Camera.h, Camera.cpp, ImagePlane.h, ImagePlane.cpp, etc. You are free to use the header files `Vector.h` and `Color.h` provided on the course webpage if you wish. The file `Vector.h` has a sufficient implementation of a linear algebra 3D vector class. Note that the specification of classes and methods here makes use of const and object references. Make sure you stick to using those. Though you may not have used them in the past very much, it is a very good habit to use them.

For the sphere, plane, and triangle classes, the method `float intersection(const Ray& r) const` computes the closest point of intersection of the object with the ray. If there is no intersection, a negative value is returned. If there is an intersection, the return value is the distance from the start of the ray to the point of intersection.

The `Scene` object is a container that can hold planes, spheres, and triangles in some way that lets the `Trace` function get to each object to determine whether the ray intersects that object, and if so, what *shaded* color the object is. There are many options for setting up such a container, all valid. You will have to select one.

You are free to create additional C++ classes, methods, etc. that have not been explicitly called for here. In fact, you will probably need to. Using std containters (e.g. vector, map, queue, etc.) is recommended wherever you see the opportunity.

You will have to create a C++ implementation file called `raytrace.cpp` that houses the main() function. You must also create a Makefile to compile and link all of the code into an executable called `raytrace`.

When executed, `raytrace` will perform the ray trace render of the required scene and write the data into an ascii ppm file. An example ppm file is provided

for you to examine. It can be viewed in any text editor to see its format, and in any image viewer to see how its format translates into an image. Do not assume that this example image looks like the assignment.

The scene you must render consists of five infinite planes, a sphere, four triangles, a camera, and a light as follows:

- The plane0 has the point $(0, 2, 0)$,the normal vector $(0, -1, 0)$, and the color $(1, 1, 1)$.
- The plane1 has the point $(0, -2, 0)$,the normal vector $(0, 1, 0)$, and the color $(1, 1, 1)$.
- The plane2 has the point $(-2, 0, 0)$,the normal vector $(1, 0, 0)$, and the color $(1, 0, 0)$.
- The plane3 has the point $(2, 0, 0)$,the normal vector $(-1, 0, 0)$, and the color $(0, 1, 0)$.
- The plane4 has the point $(0, 0, 10)$,the normal vector $(0, 0, -1)$, and the color $(1, 1, 1)$.
- The triangle0 has the vertices $(-1, 5, 0.6, 5)$, $(-1.8, 0.9, 5)$, $(-1.7, 0.4, 5)$, and the color $(245/255, 102/255, 0)$.
- The triangle1 has the vertices $(-1.8, 0.9, 5)$, $(-1.7, 0.4, 5)$, and $(1.1, 3.25, 3)$, and the color $(245/255, 102/255, 0)$.
- The triangle2 has the vertices $(-1.7, 0.4, 5)$, $(1.1, 3.25, 3)$, and $(-1, 5, 0.6, 5)$ and the color $(0, 245/255, 102/255)$.
- The triangle3 has the vertices $(1.1, 3.25, 3)$, $(-1, 5, 0.6, 5)$, and $(-1.8, 0.9, 5)$, and the color $(102/255, 0, 245/255)$.
- The sphere has the center $(1.1, 1.25, 7)$, radius 1, and the color $(0.5, 0.5, 1)$.
- The Camera has the position $(0, 0, 0)$, view direction $(0, 0, 1)$, up direction $(0, 1, 0)$, horizontal field of view 90 degrees (note that trigonometry functions take radians as input), and aspect ratio 1.3333
- The image plane has 1024 pixels horizontally (Nx) and 768 pixels vertically (Ny).
- The point light is at position $(-1, -1, 7)$ and has color $(1, 1, 1)$.

Description for 6050 Students

In addition to the description above, you must render a second image after adding a second point light to the scene. The second light affects only the triangles and the sphere. The second light has position $(1, 1, -3)$ and color $(0.15, 0.35, 0.05)$.

Upload to handin

Create a folder called `<username>`. Put all of the following files into that folder. There should be no subfolders.

```
Makefile
Vector.h
Camera.h
ImagePlane.h
Ray.h
Trace.h
Sphere.h
Plane.h
Triangle.h
Light.h

Camera.cpp
ImagePlane.cpp
Ray.cpp
Trace.cpp
Sphere.cpp
Plane.cpp
Triangle.cpp
Light.cpp

raytrace.cpp

output.ppm

(any other files you need to include)
```

If you are in 6050, you will also need to include:

```
output2.ppm
```

Zip compress the folder into a single zip file, named `<username>.zip`. Upload this file to the handin system. The course webpage has more guideance and caveats if you need them.