CS130 - Ray Tracer Quickstart Guide

Compile command: `scons`

Run test 01 for homework 2: `./ray_tracer -i tests-hw2/01.txt`

Running the program on a test file will generate an image file `output.png`, which is the render your program made with the given test parameters.

Compare test 01 for homework 2 with its solution image:
`./ray_tracer -i tests-hw2/01.txt -s tests-hw2/01.png`

Run grading script on the tests for homework 2: `python ./grading-script.py tests-hw2`

Functions to implement for this lab:

- [ ] `camera.cpp`: `World_Position`
- [ ] `render_world.cpp`: `Render_Pixel`; (only ray construction)
- [ ] `render_world.cpp`: `Closest_Intersection`
- [ ] `render_world.cpp`: `Cast_Ray`
- [ ] `sphere.cpp`: `Intersection` (returns intersection of ray and the sphere.)
- [ ] `sphere.cpp`: `Normal` (returns the normal vector at the point hit.)
- [ ] `plane.cpp`: `Intersection` (returns intersection of ray and the plane.)

Important Classes

- `render_world.h/cpp`: class `Render_World`. Stores the rendering parameters such as the list of objects and lights in the scene.
- `camera.h/cpp`: class `Camera`. Stores the camera parameters, such as the camera position
- `hit.h`: class `Hit`. Stores the ray object intersection data such as the distance from the endpoint to the intersection point with the object.
- `ray.h`: class `Ray`. Stores ray parameters: `end_point`, `direction`. `vec3 Point(double t);` returns the point on the ray at distance t.
- `sphere.h/cpp`: class `Sphere`. Stores sphere parameters (center, radius).
- `plane.h/cpp`: class `Plane`. Stores plane parameters (x0, normal).

The general algorithm for ray tracing is as follows:

```
for all pixels (i, j) do
    Compute the "world position" of the pixel.
    Create a ray r from the camera position to the world position of the pixel
    Find the closest object o that intersects with the ray.
    if o = ∅ then
        Use background_shader.
    else
        Use shader associated with o.
    Get the pixel color c by using the SHADE_SURFACE function of the shader.
```

World position of a pixel (`camera.cpp`). The world position of a pixel can be calculated by the following formula: $\mathbf{p} + C_x\mathbf{u} + C_y\mathbf{v}$, where $\mathbf{p}$ is `film_position` (bottom left corner of the screen), $\mathbf{u}$ is `horizontal_vector`, $\mathbf{v}$ is `vertical_vector`, and C is the `vec2` obtained by `Cell_Center`(pixel_index); see `camera.h`.

Constructing the ray (`Render_Pixel` function). `end_point` is the camera position (from camera class). `direction` is a unit vector from the camera position to the world position of the pixel. Note that `vec3` class has a `normalized()` function that returns the normalized vector.

`Closest_Intersection`.

```
procedure CLOSEST_INTERSECTION
    Set min_t to a large value (google std::numeric_limits)
    for all shaded objects o do
        Use o.object->Intersect to get the closest hit with the object
        if Hit is the closest so far and larger than small_t then
            Store the hit as the closest hit
    return object hit and closest hit
```

`Cast_Ray`. Get the closest hit with an object using `Closest_Intersection`. If there is an intersection set color using the associated shader `Shade_Surface` function which calculates and returns the color of the ray/object intersection point. `Shade_Surface` receives as parameters: the `render_world` that called it, ray, hit, intersection point, normal at the intersection point and recursion depth. You can get the intersection point using the ray object and the normal using the object pointer inside the `Shaded_Object`. If there is no intersection, use `background_shader` of the `render_world` class. The background shader is a `flat_shader` so you can use any 3d vector as parameters. If there is no background shader, return black.