CS130 - LAB - Raytracing Project

Name: Zifei Wu SID: 862255054

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 material_shader on o.
    Get the pixel color c by using the SHADER_SURFACE function of the shader.
```

1. Please find the appropriate files in the skeleton code and fill the blanks below.

(a) The `World_Position` function in the Camera class returns the world position of a given pixel (`ivec2 pixel_index`). This function is implemented in `camera.cpp` starting from line # 42 .

(b) The `Cell_Center` function in the Camera class returns the screen position of a given pixel, `ivec2 pixel_index`. This function is implemented in `camera.h` starting from line # 54 .

(c) Locate where the loop that iterates through all pixels. The loop is located in function Render in `render_world.cpp`

(d) The `Cast_Ray` function in `render_world.cpp` returns the color of the pixel using the shader of the closest object it intersects with. Find the function in `render_world.cpp` and fill below. `Cast_Ray` function is implemented in `render_world.cpp` starting from line # 49 . `Cast_Ray` function is called in the function Render_Pixel. in `render_world.cpp`.

(e) `Closest_Intersection` function will be used in `Cast_Ray` function to find the closest object that intersects with the ray and (if any) provide it's intersection information in a object of type Hit. The `Closest_Intersection` function is implemented in render_world.cpp starting from line # 25 . The output object hit should store the following information: object, distance along the ray, location of hit, rotation, whether exiting. Any intersection with distance $\leq$ small_t should be ignored .

(f) The Intersection function is a function of the Object class (object.h) which is a base class for scene objects such as Plane and Sphere. This function is overloaded by these classes. It populates the std::vector<Hit>& hits argument with a list of all intersections between the ray and object (including the beginning of the ray if it is inside) in order along the ray. When there is no intersection, the caller can determine this by checking whether the size of hits is 0 .

2. Write C++ code using vec.h to accomplish each of these tasks. You may assume that $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ are of type vec3 and that a and b are scalars of type double.

(a) $\mathbf{u} = (2, 3, 5)$

vec u(2,3,5);

(b) $\mathbf{w} = \frac{\mathbf{u}}{a} + \frac{3}{b}\mathbf{v}$

w= (u/a)+ v*(3/b);

(c) $\mathbf{w} = 3\mathbf{u} \times \mathbf{v}$

w= cross (u*3, v);

(d) Normalize $\mathbf{u}$ in place.

u = u.normalized();

(e) $\mathbf{w} = (\|\mathbf{u}\| + 1)\mathbf{v}$.

w = v * (u.magnitude() +1);

(f) $a = \frac{1}{4}\mathbf{u} \cdot \mathbf{v}$

a= dot (u·(1/4), v);

(g) $a = \mathbf{u}_0$ (get the first entry from the vector)

a= u₀.x[0];

Getting started with the ray tracer project

Compile command: scons

Run test 05: `./ray_tracer -i 05.txt`

Compare test 05: `./ray_tracer -i 05.txt -s 05.png`

Run grading script (note the extra period): `./grading-script.py .`

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.)
- ☐ `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).

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 objects o do
        Use o->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 closest hit
```

`Cast_Ray`. Get the closest hit with an object using `Closest_Intersection`. If there is an intersection set color using the object `Shade_Surface` function which calculates and returns the color of the ray/object intersection point. `Shade_Surface` receives as parameters: ray, 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 hit 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.