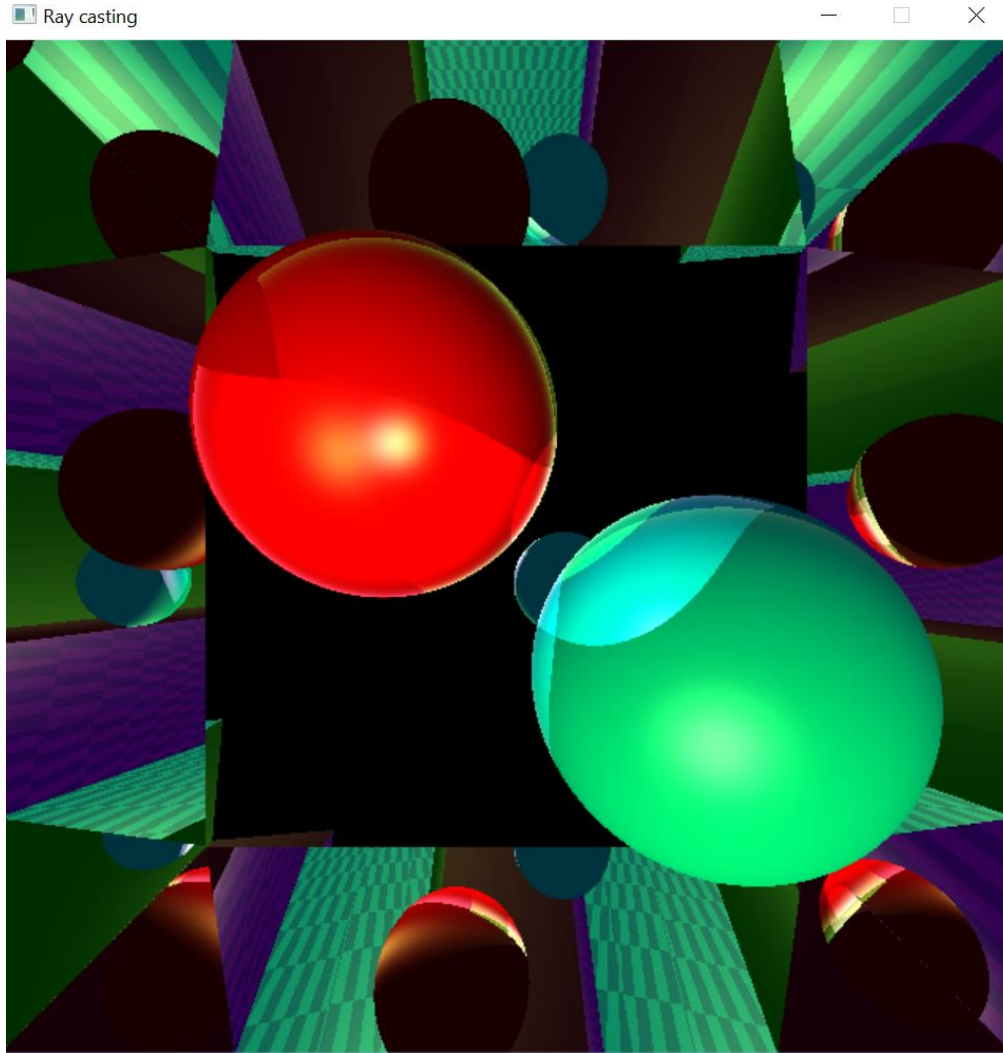


Assignment 2 – Ray Tracing



Overview

The concept of ray tracing: a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects. The technique can produce a very high degree of visual realism, usually higher than that of typical scan line rendering methods, but at a greater computational cost.

The objective of this exercise is to implement a ray casting/tracing engine. A ray tracer shoots rays from the observer's eye through a screen and into a scene of objects. It calculates the ray's intersection with objects, finds the nearest intersection and calculates the color of the surface according to its material and lighting conditions. **(This is the way you should think about it – this will help your implementation).**

Requirements

Read this entire explanation before starting. Understand the slides taught in class especially.

The feature set you are required to implement in your ray tracer is as follows:

- Background
 - Plain color background
- Display geometric primitives in space:
 - Spheres
 - Planes
- Basic lighting
 - Directional lights
 - Spot lights
 - Ambient light
 - Simple materials (ambient, diffuse, specular...)
- Basic hard shadows
- **Reflection**, up to 5 recursive steps (mirror) after 5 steps take the material properties of the last mirror the ray hits.
- Sphere **transparency** (according to snell's law, only spheres can be transparent), up to 5 recursive steps (mirror) after 5 steps take the material properties of the last object the ray hits.
- Multi-sampling for anti-aliasing (5 bonus points)

Shader

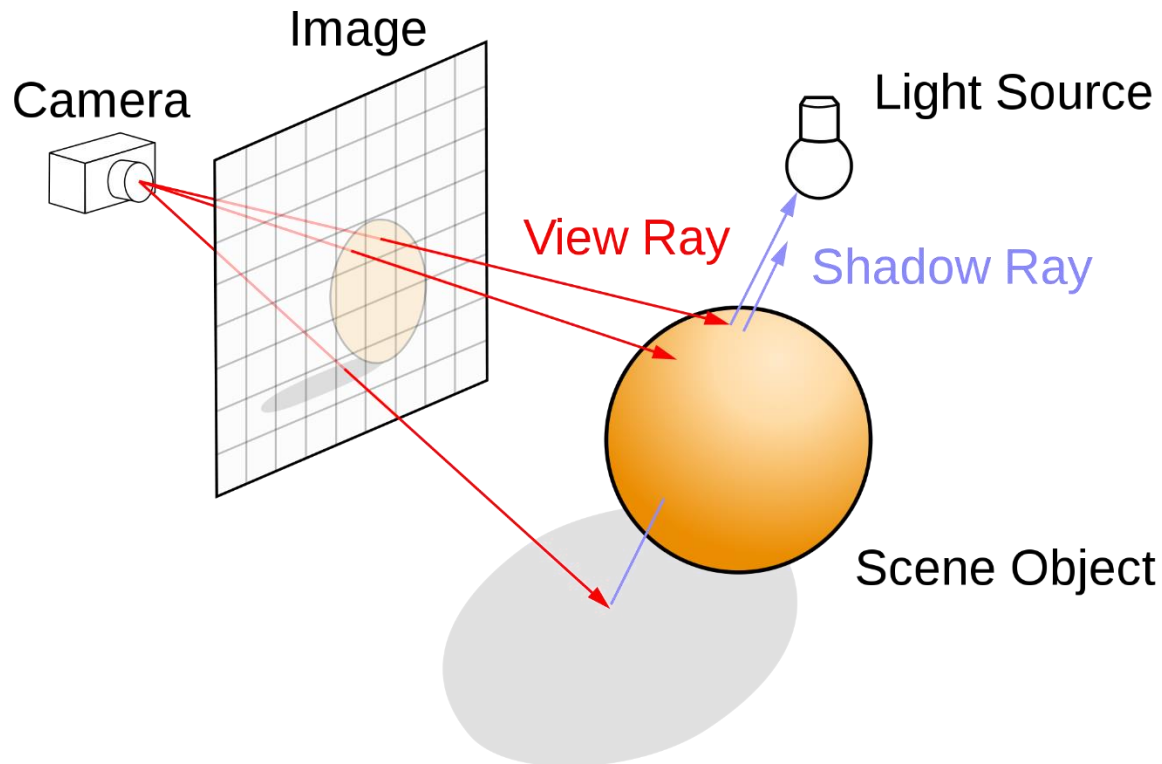
You will implement ray tracing on the fragment shader. Template fragment shader is supported in assignment page. **Remember that recursion is not allowed in shading language.**

Scene Definition

The 3D scene for rendering will be defined in a scene definition text file. The scene definition contains all parameters required to render the scene and the objects in it. The specific language used in the definition file is defined later.

Screen

The screen is located on $z=0$ plane. The right up corner of the screen is located at $(1,1,0)$ and the left bottom corner of the screen is located at $(-1,-1,0)$. All in the scene coordinates.



Geometry

Geometric primitives

Sphere

A sphere is defined by a center point and scalar radius. The normal of each point on the sphere's surface is easily computed as the normalized subtraction of the point and the center.

Plane

A plane is defined by a normal to the plane and a negative scalar which represents the "d" in the plane equation ($ax + by + cz + d = 0$). Every plane is an infinite plane and will be divided to squares in checkers board pattern. In dark squares the diffuse component of Phong model has 0.5 coefficient.

Hint:

```
(mod(int(1.5*p.x),2) == mod(int(1.5*p.y),2))
```

Spheres and plane may intersect each other.

Background

The background of the rendered scene is a flat color.

Camera

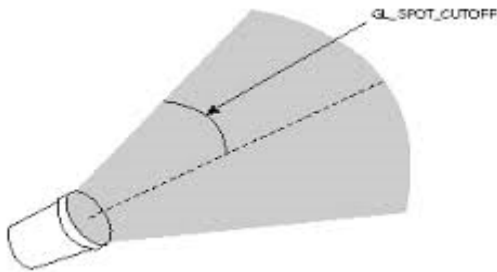
The camera is a simple pinhole camera as described in the lecture slides and will be located at eye coordinates (you will get as input) and looks towards the center of the screen.

Lighting

Basic lights and shadows

For basic lighting you need to implement:

1. Ambient lighting – a color that reflects from all surfaces with equal intensity.
2. Directional – A light source like the sun, which lies at infinity and has just a direction.
3. Spot light – A point light source that illuminates in a direction given by a direction vector . This light source has a cutoff angle as describe in the following image.



Every light source has its own intensity (color) and there can be multiple light sources in a scene. Shadows appear where objects obscure a light source. In the equation they are expressed in the S_i term. To know if a point p in space (usually on a surface) lies in a shadow of a light source, you need to shoot a ray from p in the direction of the light and check if it hits another object. If it does, make sure that it really hits it before reaching the light source and that the object which is hit is not too close (or it is not the same object) .

Materials

You need to implement the lighting formula (Phong) from the lecture slides. The material of a surface should be flat with ambient, diffuse, specular reflections, and a shininess parameter (the power of $V \cdot R$ when V and R are unit vectors).

Shadow

You will add shadow effect to your scene using S_i parameter in Phong model (see lecture slides). Shadow appears when some object located between the light source and another object which can be observed by the viewer. In that case you should ignore the covered light source part in Phong model. Some common mistakes may cause spurious shadows to appear. Make sure you understand the vector math involved and all the edge-cases.

Reflection

You can sign a mirrored object by "r". Each ray which hits a mirror object breaks symmetrically to the normal. The color of the pixel in a mirror object is calculating according to the breaking ray. Don't forget to consider the light sources rays which break by the mirror objects. You can

ignore mirror objects material parameter (ambient, diffuse, specular) or you can use it to color the reflect light.

Mouse input (like in the previous assignment)

The user can move the camera (eye) up, down, left and right when pressing mouse left button. Move the camera closer and farther from the scene when the user scroll up and down.

Transparent spheres

Use Snell's law (see lecture 3 slides 73-74) with refractive index of 1.5 to determine the direction of the light rays intersect with the sphere (air have refractive index of 1)

Multi-Sampling (5 bonus points)

Shoot 4 or more rays from each pixel and combine the color values you get to determine the pixel color.

Input

You will get one text file named scene.txt (see example).

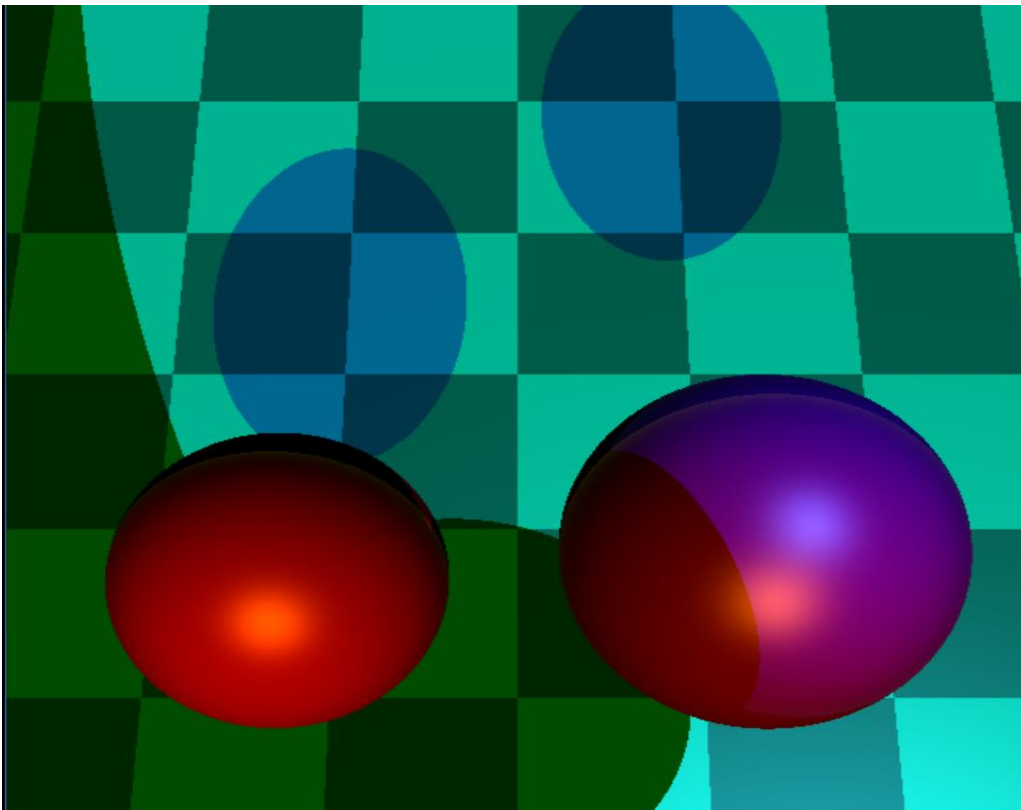
- The first line in the file will start with the letter "e" followed by initial camera (eye) coordinates. The fourth coordinate can use to transfer additional data (if you choose to implement a bonus you can use it as bonus mode flag).
- Second line starts with "a" followed by (R,G,B,A) coordinate represents the ambient light.

From the third row we will describe the object and lights in scene:

- Light direction will describe by "d" letter followed by light direction (x, y, z, w). 'w' value will be 0.0 for directional light and 1.0 for spotlight.
- For spotlights the position will appear afterwards after the letter "p" (x,y,z,w). 'w' value will be the cosine of the Cutoff angle.
- Light intensity will describe by "i" followed by light intensity (R, G, B, A).
- Spheres and planes description will appear after "o", "r" or "t". For spheres (x,y,z,r) when (x,y,z) is the center position and r is the radius (always positive, greater than zero). For planes (a,b,c,d) which represents the coefficients of the plane equation when 'd' gets is a non-positive value.
- "r" represents reflective object and "t" represents transparent object.
- The color of an object will appears after "c" and will represent the ambient and diffuse values (R,G,B,A), when 'A' represents the shininess value.
- **The specular value of an object is always (0.7,0.7,0.7).**

Input example

e 0.0 0.0 4.0 1.0
a 0.1 0.2 0.3 1.0
o 0.0 -0.5 -1.0 -3.5
o -0.7 -0.7 -2.0 0.5
o 0.6 -0.5 -1.0 0.5
c 0.0 1.0 1.0 10.0
c 1.0 0.0 0.0 10.0
c 0.6 0.0 0.8 10.0
d 0.5 0.0 -1.0 1.0
d 0.0 0.5 -1.0 0.0
p 2.0 1.0 3.0 0.6
i 0.2 0.5 0.7 1.0
i 0.7 0.5 0.0 1.0



Moving the camera and the objects in the scene

Key input

Change `key_callback` function in `inputManager.h`

- 'up arrow' rotates the camera toward its up direction.
- 'down arrow' rotates the camera toward its minus up direction (down).
- 'left arrow' rotates the camera toward its minus right direction (left).
- 'right arrow' rotates the camera toward its right direction.
- The distance of the camera from the center of the screen will always stay the same. You may use spherical coordinates representation (see PS3 slide 25)

Mouse input

Change `mouse_callback`, `cursor_position_callback`, `scroll_callback` functions in `inputManager.h`

The user can move the camera (eye) up, down, left and right when pressing mouse left button. Move the camera backward and forward when the user scroll up and down.

The user can pick an object by press the mouse right button (and hold it). If the picked object is a sphere move it according to the mouse on the camera plane (the plane that is spanned by up and right vectors), else rotate the chosen plane around the up vector (when the mouse moves from left to right or vice versa), or around the right vector (when the mouse moves from down to up or vice versa)

Notes

Getting started

- Design Before you start coding! You should think carefully about the structure and design of your Shader. Start from a small scene with one or two objects.
- The basis for your code on glfw2D code which appears in useful links page.

Validation

We will provide you with sample scenes for validation and the way they are supposed to render in your ray tracer. Your implementation may vary from the supplied image in small details but in general the scene should look the same.

Some general hints:

- Before plotting a pixel, make sure that its color does not exceed the range of 0-1 for every color channel (you can use `max`, `min` or `clamp` functions).
- You can debug shaders by coloring pixels differently or you can implement problematic parts in C++ and then copy the implementation to the fragment shader.

- Don't use "==" for comparing floats

Submission:

Submission is in pairs.

Submit inputManager.h ,main.cpp, Assignment2.cpp, Assignment2.h and other new files you have created for the assignment. Submit your shaders (also the vertex shader). **Don't change engine files that don't mention here.** Zip everything to <ID1>_<ID2>.zip file.

Helpful links:

[http://en.wikipedia.org/wiki/Ray_tracing_\(graphics\)](http://en.wikipedia.org/wiki/Ray_tracing_(graphics))

<http://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html>

Good Luck!