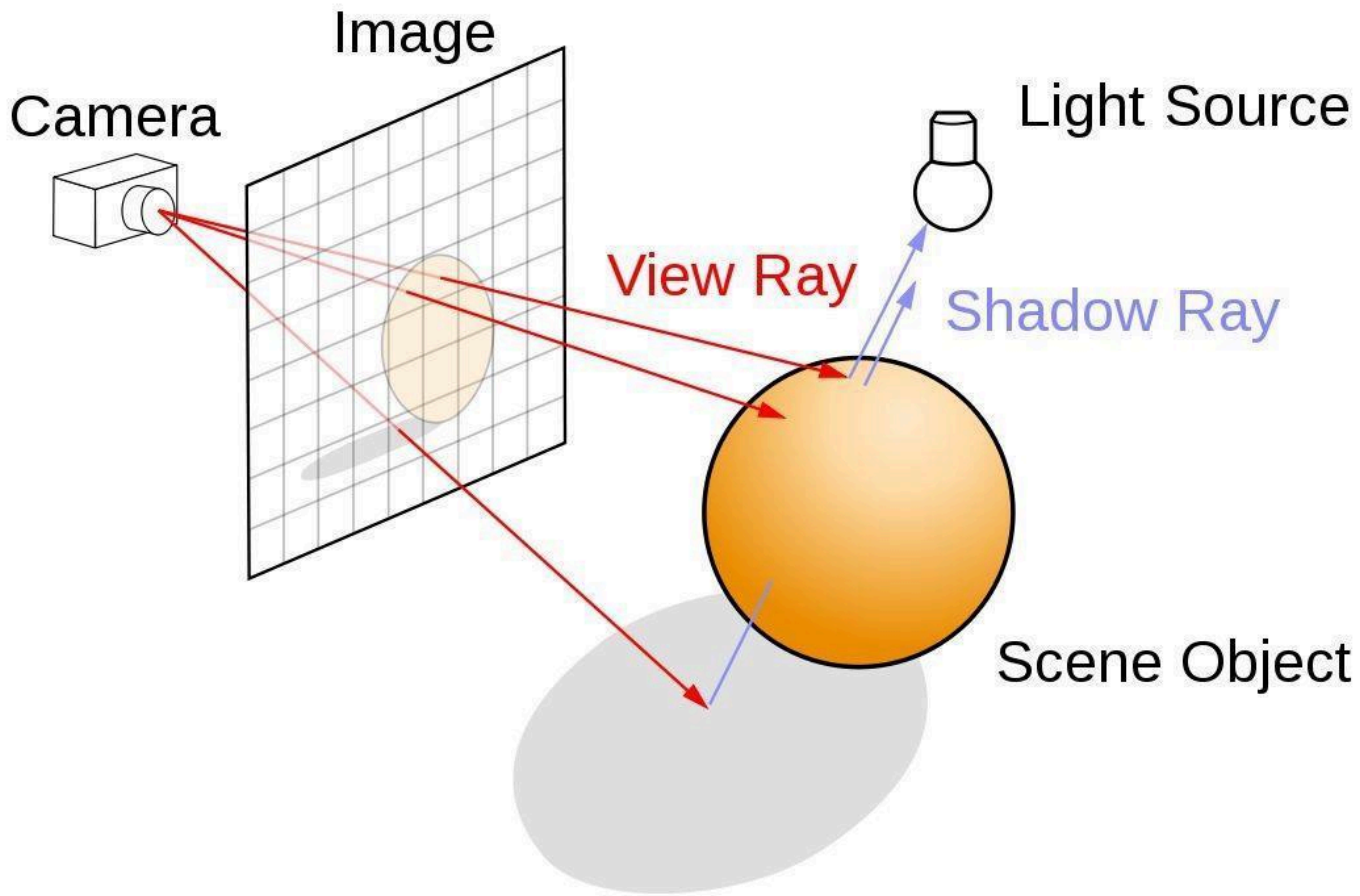
Assignment 2 – Ray Tracing



Overview:

Ray Tracing is 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 manually:

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 the scene 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).

Part 1: C++ RayTracer

Requirements:

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

- Display Geometric data in the 3D space:
  - Spheres
  - Planes
  - Background
- Basic Light Sources:
  - Global Ambient light
  - Directional lights
  - Spotlights
- Basic Materials color (Ambient, Diffuse, Specular)
- Basic Hard Shadows
- Reflection, up to 5 recursive steps (mirror object)
- Transparency (Spheres Only), up to 5 recursive steps (crystal ball)

The resulting image should be saved to a **.png** file with the **stb** library.

Scene Definition:

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

Screen and Camera:

Screen and Camera Should be implemented in 2 Phases
(i.e, implement the entire scene components with Phase 1, and the advance to phase 2):

- **Phase 1 - Fixed Camera:**
  - Screen located on the XY Plane, at $Z = 0$.
  - Right up corner of the screen is located at $(1, 1, 0)$ in the 3D scene.
  - Left up corner of the screen is located at $(-1, -1, 0)$ in the 3D scene.
  - Camera position specified in input file.
  - The camera looks towards the center of the screen.
- **Phase 2 – Free Camera:**
  - Camera position, V_{up} and V_{to} are specified in the input file.
  - Screen width, height, and distance from camera are specified in the input file.
  - Screen plane is orthogonal to V_{to}, and aligned with V_{up} in the 3D scene.

Geometric Primitives:

- **Sphere:** defined by a center point (x, y, z) and scalar radius r.
- **Plane:** defined by an un-normalized normal vector (a, b, c) to the plane and a negative scalar which represents the d in the plane equation:
  $(ax + by + cz + d = 0)$
  Don't forget to normalize (a, b, c) for intersection calculation after finding Q_0.

  Every plane is an infinite plane and will be divided into squares in checkerboard pattern.
  In dark squares the diffuse component of the Phong model has 0.5 coefficient.

Spheres and Planes may intersect with each other.
You may use the following code to get the checkerboard pattern:

```
vec3 checkerboardColor(vec3 rgbColor, vec3 hitPoint) {
	// Checkerboard pattern
	float scaleParameter = 0.5f;
	float checkerboard = 0;
	if (hitPoint.x < 0) {
		checkerboard += floor((0.5 - hitPoint.x) / scaleParameter);

	else {
		checkerboard += floor(hitPoint.x / scaleParameter);
	}
	if (hitPoint.y < 0) {
		checkerboard += floor((0.5 - hitPoint.y) / scaleParameter);
	}
	else {
		checkerboard += floor(hitPoint.y / scaleParameter);
	}
	checkerboard = (checkerboard * 0.5) - int(checkerboard * 0.5);
	checkerboard *= 2;
	if (checkerboard > 0.5) {
		return 0.5f * rgbColor;
	}
	return rgbColor;
}
```

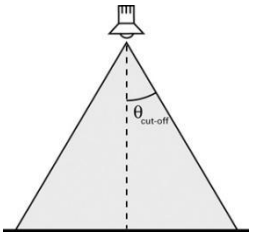
Background:

Background of scene is black color $(0, 0, 0)$.

Basic Light Sources:

Three types of light:

- **Global Ambient light** – A light that reflects from all surfaces with equal intensity
- **Directional light** – A light source like the sun, which lies at infinity and has just a direction.
- **Spotlight** – A point light that illuminates in a direction given by a direction vector.
  This light source has a cutoff angle as described in the following image.



Basic Materials:

You need to implement the lighting formula for the Phone model.
The material of a surface should be flat with Material **Ambient**, **Diffuse**, **Specular** colors, and Material **Shininess** (Phong exponent).

Basic Hard Shadows:

You will add a shadow effect to your scene using parameters in the Phong model.
Shadow appears when some object is located between the light source and another object which can be observed by the viewer (or outside the cutoff angle of spotlight sources).
In that case you should ignore the covered light source part in the Phong model.
Some common mistakes may cause spurious shadows to appear.
Make sure you understand the vector math involved and all the edge-cases.

Notice: Spotlight sources have a position, and objects can appear behind them.
When you are trying to find if another object might block the spotlight source, do not consider objects that are behind the spotlight.

Reflection:

Each ray which hits a mirror object breaks symmetrically to the normal.
The color of the pixel in a mirror object is calculated according to the breaking ray.
For the reflective objects, ignore the material parameter (Ambient, Diffuse, Specular) and use the color from the reflected light.

Transparency (Spheres Only):

Use Snell's law with a refractive index of 1.5 to determine the direction of the light rays that intersect with the sphere (air has refractive index of 1).
For the transparent objects, ignore the material parameter (Ambient, Diffuse, Specular) and use the color from the refracted light.

Notice: You may assume that there will be no other objects that intersect or appear inside the transparent objects (Since it will require handling a lot of edge cases), and raise the recursive count after the ray gets out of the sphere.

Input:

You will get scene text files (like in the example), where the first parameters meaning are as follows:

- **"e"** (eye) – Represents the **Camera position** coordinates (x, y, z, d).
  The 4th coordinate is the distance to screen (can be ignored in Phase 1).
- **"u"** (up) Represents the camera's up vector (x, y, z, h).
  The 4th coordinate is the height of the screen in the 3D scene.
- **"f"** (forward) Represents camera's direction towards the screen (x, y, z).
  The 4th coordinate is the width of the screen in the 3D scene.
- **"a"** (ambient) – Represents the **Global Ambient Intensity** (r, g, b).
  The 4th coordinate will always be 1.0 and can be ignored.

Notice: "u" and **"f"** parameters can be ignored in Phase 1.

The next parameters represent the information about the **light sources** and **objects**:

- **"d"** (direction) – Represents the **Light source direction** (r, g, b).
  The 4th coordinate value will be 0.0 for **Directional light** and 1.0 for **Spotlight**.
- **"p"** (position) – (Only for spotlights) Represents the **Spotlight** position coordinates (x, y, z).
  The 4th coordinate value represents the **cutoff angle cosine value**.
  ("**p**" order corresponds to the **"d"** spotlights order)
- **"i"** (intensity) – Represents the **Light source Intensity** (r, g, b).
  The 4th coordinate will always be 1.0 and can be ignored.
  (**"i"** order corresponds to the **"d"** order)
- **"o"** (object), **"r"** (reflective) or **"t"** (transparent) – Represents Spheres and Planes, where:
  - Spheres (x, y, z, r) – where (x, y, z) is the center position and **r** is the radius $(r > 0)$.
  - Planes (a, b, c, d) – where (a, b, c, d) represents the coefficients of the plane equation $(d \leq 0)$.

  Notice the following things:
  - The 4th coordinate determines if the object is a sphere or a plane.
  - Spheres and Planes can be either **normal** objects, **reflective** objects or **transparent** objects and will require different handling based on their type.
  - For "r" (reflective) and "t" (transparent) and next "c" parameter values can be ignored.
- **"c"** (color) - Represents the **Ambient Material** and **Diffuse Material** color (r, g, b).
  The 4th coordinate represents the **Shininess** value.
  (**"c"** order corresponds to the **"o"**, **"r"**, **"t"** order)

The input parameters explanation according to the Phong model equation:

In this Assignment, this is the parameters corresponding values (per object on the intersection point):

I_E – Assume as $(0, 0, 0)$ for all objects.
I_A – The (r, g, b) values of "a".
l_i – The (r, g, b) values of the i-th "i".
I_R – Assumed as $(0, 0, 0)$ for normal object "o" and $(1, 1, 1)$ for reflective "r" and transparent "t" objects.
K_A – The (r, g, b) values of "c".
K_D – The (r, g, b) values of "c".
K_S – Assume as $(0.7, 0.7, 0.7)$ for all objects.
K_R – The final color from the recursive calculations of reflective or transparent objects.
S_i – The shadow term binary value (0 if the i-th light source is blocked, 1 otherwise)
n – The 4th value of "c".
Normalized vectors – Use the values of the remaining parameters to calculate them as required

Color Model Summary

- I_E – Material Emission
- I_A – Global Ambient
- I_i – Light Source i Intensity
- I_R – Reflection Intensity
- K_A – Material Ambient
- K_D – Material Diffuse
- K_S – Material Specular
- K_R – Material Reflection
- S_i – Light Shadowed?
- n – Material Shininess
- Normalized vectors:
  - L – Intersection to light
  - N – Intersection normal
  - V – Intersection to Eye
  - R – Intersection to reflected light

$$I = I_E + K_A I_A + \sum_i (K_D (N \cdot L_i) + K_S (V \cdot R_i)^n) S_i I_i + K_R I_R$$

Notes:

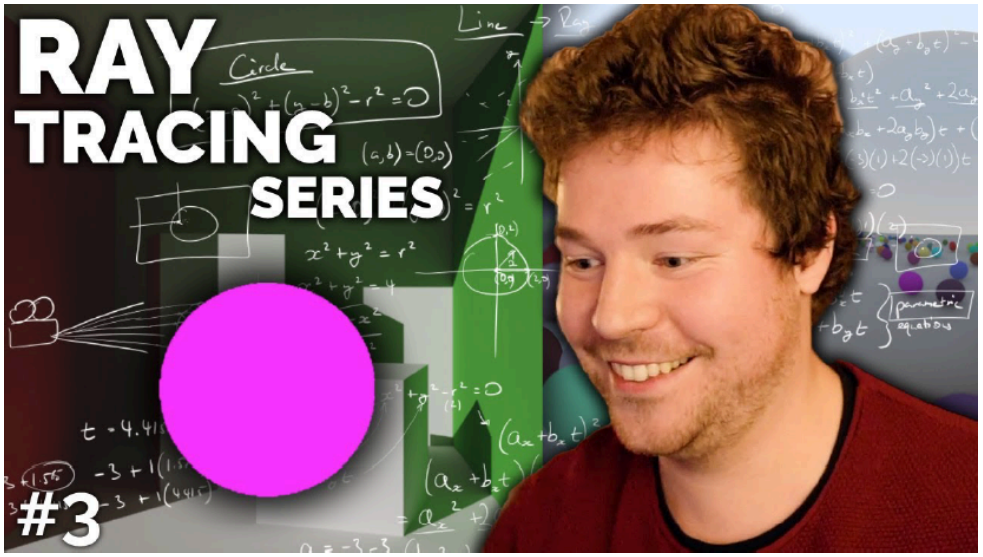
Design Before you start coding!
Think about how you will represent the scene information in a way that will be easy for you to work with.
Start checking from a small scene with one or two objects.

Helpful links:

https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

https://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html

You can watch the ray tracing series as well (The image is clickable).



How to Start:

Start implementing your solution in the following order, and validate in each step that you get the expected results:

1. Design the system: Define the data structures you will use to store the relevant data from the text file. You may find it helpful to use Object Oriented Programming and build classes and hierarchies for the **Objects** and **Light sources**
   (For example: A Plane is an Object; Directional light is a Light source), and also for the Rays and Intersection Points to hold your current data during a recursive call.
2. Build the reader of the text scene data, that loads the information to the data structures you have defined on step 1.
3. Define the Camera and screen according to **Phase 1** and build the double for loops for running over all the pixels on the screen and finding the rays that pass through them. To do that, find the sizes of a single pixel and calculate a pixel center to find the ray that passes through it.
4. (The important part) Define a simple sphere or a box of your own and check that with the rays that you have built for each pixel you can get the sphere on the screen like in the provided video above.
5. Replace the sphere you have built with the spheres data from the text file you have read and check that you can see them on the scene.
6. Now you can move to implement the full Phong equation. Start with calculating the Ambient, Diffuse and Specular color of a Normal object.
7. Move to Phase 2, and implement a free camera.
8. Implement the Hard Shadows and finally move to the Reflective and Transparent objects.

General tips:

- Before plotting a pixel, make sure that its color does not exceed the range of 0-1 (or 0-255) for every color channel.
- In the Transparency part, when trying to find the second intersection point inside the sphere you might get that there are 2 intersection points on the ray. In that case, always take the further intersection point (This might be happening when due to numerical errors, the closer point will have very small positive t value instead of absolute zero).
- The **Right** vector of the screen will be orthogonal to the **Up** vector and the **Toward** vector.

Validation:

We will provide you with sample scenes for validation and the way they are supposed to be rendered in your ray tracing algorithm.
Your displayed result may vary from the supplied image in small details, but in general the scenes should look the same.

10 pts bonus - Ray Tracing, Sampling and Reconstruction:

Define a checkerboard scene only (no spheres).
Adjust the checkerboard size (0.5 scaleParameter in code now)

1. Raytrace the scene
2. Compute the FFT of the resulting image (use FFTW for performing the 2D Discrete Fourier Transform (DFT) in C++.
   The primary functions for this are fftw3::fftw_plan_dft_r2c_2d() and fftw3::fftw_execute()
3. Define the max sampling rate based on FFT max frequency (nyquist).
4. Raytrace scene at given sampling rate
   (Think of how many rays are needed to be shot according to sample rate).
5. Compute the FFT of the resulting image.
6. Compare.

Part 2: Fragment Shader RayTracer

In this part you will implement a real-time fragment shader!

Requirements:

Requirements for this assignment are similar to Part 1, but some are not mandatory:

- Display Geometric data in the 3D space:
  - Spheres
  - Planes
  - Background
- Basic Light Sources:
  - Global Ambient light
  - Directional lights
  - Spotlights
- Basic Materials color (Ambient, Diffuse, Specular)
- Basic Hard Shadows – (**Bonus**)
- Reflection, up to 5 recursive* steps (mirror object) (**Bonus**)
- Transparency (Spheres Only), up to 5 recursive* steps (crystal ball) (**Bonus**)

* Recursion is not supported by GLSL. See notes below.

The bonus part is 10 pts in total.

Structure:

Project repo can be found in this link: https://github.com/yachilT/WebGL-Raytracer

It contains shaders\raytracer.frag – an incomplete fragment shader written in glsl with empty functions.

Your task is to implement those functions.

Feel free to add any helper method and structs.

Implemented Structs:

- Camera – position, forward, right, up. All vec3
- Plane – has point, normal (normalized), color. All vec3
- Sphere – has center, radius, color, type.
  Type acts as enum with 3 values: opaque, reflective and refractive.
- Light – Represents both directional and spotlight.
  If the cutoff is positive, light is spotlight and position may be used. Otherwise it is a directional light (and position is ignored). Cutoff value is like in Part 1 (cutoff angle cosine value).

Implemented Objects:

- uSpheres – An array of spheres with length uNumSpheres.
- uLights – An array of lights with length uNumLights.
- uPlane – The plane.

How to run:

Assuming you have python installed on your machine, open terminal in the same directory of the project and run:

1. python -m http.server 8080 (It will open a local server on your machine)
2. open your browser and navigate to http://127.0.0.1:8080/
3. Done.

How to interact:

- Click on the screen to enter the scene.
- Use WASD keys to move from back left and right.
- Use space and shift to move up and down.
- Use your mouse to look around.
- Press P for the scene itself to move!

Notes:

- Since there isn't inheritance, you'll need to handle intersections of spheres and the plane separately.
- For transparency and reflection, recursion is not supported.
  You'll need to implement an iterative version of the recursion (loop instead of function call).
- You should define a hit info struct for your convenience.

Assignment Score:

- The cpp ray tracer: **70 pts**
- Shader real time ray caster: **30 pts**
- Checkerboard FFT sampling and reconstruction: **10 pts bonus**
- Shader shadows, refraction, reflection: **10 pts bonus**
- Note: Assessment includes both your produced results and your demonstrated understanding of the code, technical concepts, and theoretical principles.

Submission:

Submit zip file with the following files:

1. Link to your **GitHub** repository. It should contain both **Part 1 & Part 2**
2. **(Optional)** Text, Doc or PDF file with short explanation about the changes you did in the engine (files you change and functions you modify).
   This file will help you and us to understand what changes you have made in the engine to complete the assignment 😊

The zip file name must include your ID numbers as follows: **ID1_ID2.zip**.

Good Luck!