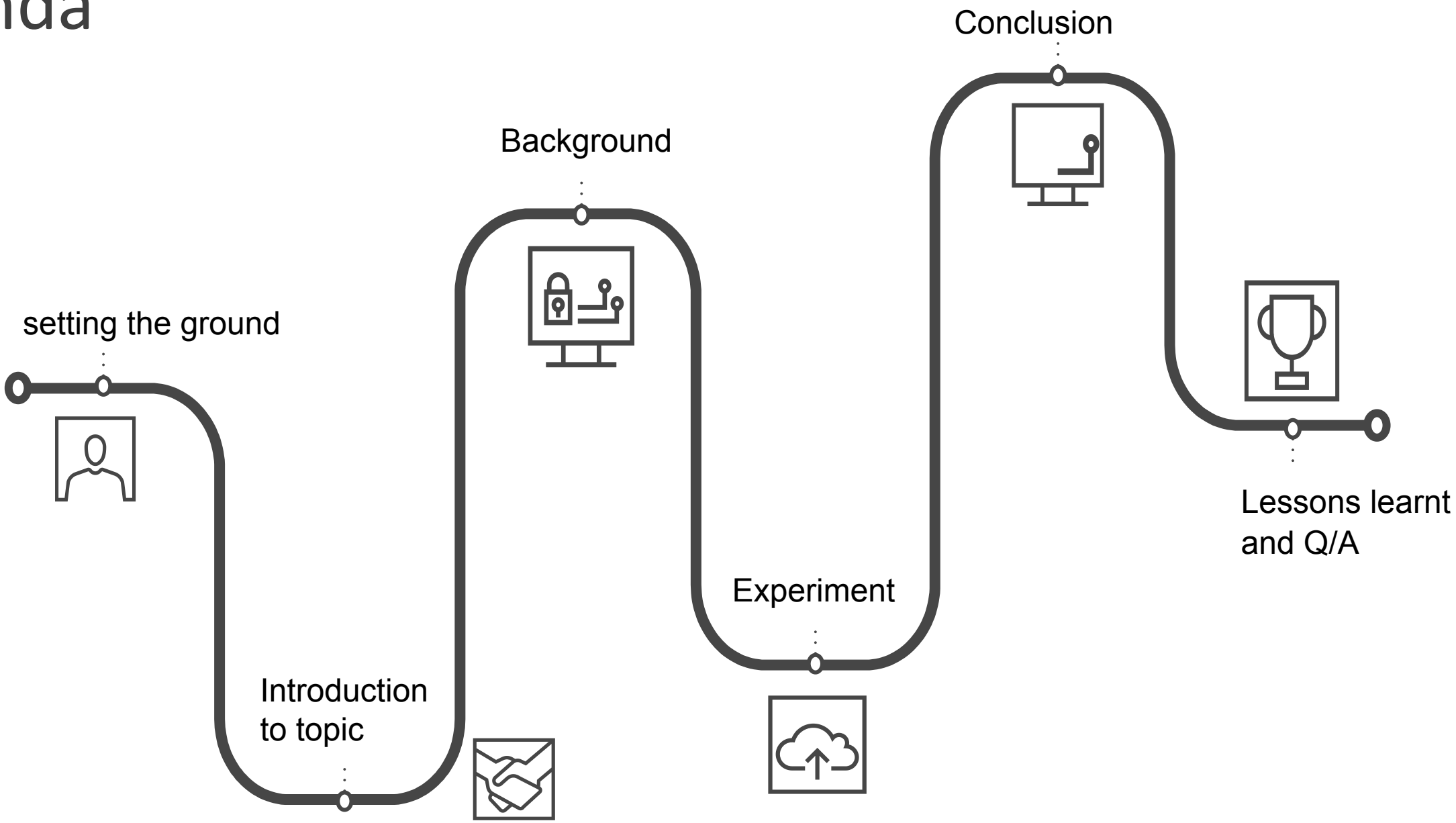
Parallel Ray Tracing

Seminar with Practical: Scalable Computing Systems and Applications in AI, Big Data and HPC

Supervisor: Zoya Masih
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Submitted by:
Karan Sharma
Pranay Bhatia

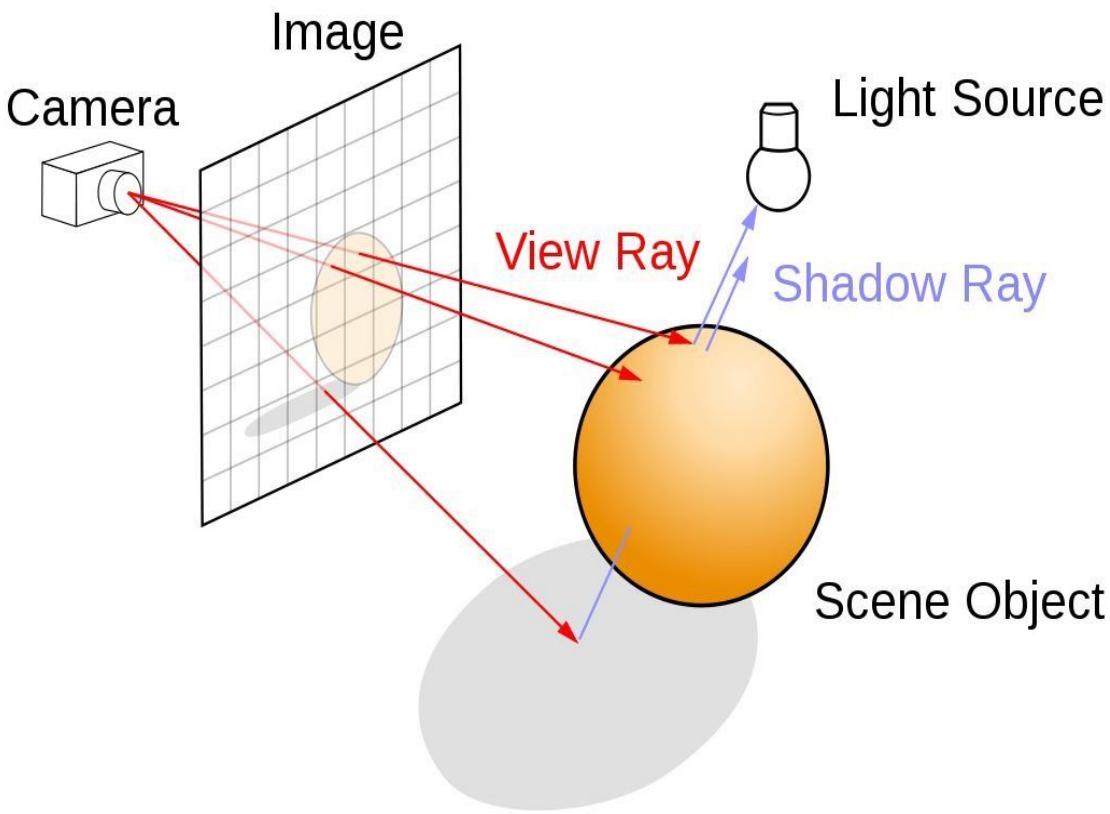
Agenda


setting the ground
Introduction to topic
Background
Experiment
Conclusion
Lessons learnt and Q/A

Introduction

- The **objects** you're seeing are **illuminated by beams of light.**
- Follow the path of those beams backwards from your eye to the objects that light interacts with.
- That's ray tracing.

Ultimate Goal: Produces images that can be indistinguishable from those captured by a camera**.** Even in an animated film.



https://link.springer.com/article/10.1007/s11227-021-03680-0

Introduction

Technique in modern movies rely on to generate or enhance special effects

- reflections
- refractions
- shadows

Uses:

- Non-real-time (film and television)
- Real-time (video games)
- Lightning design
- Architecture and engineering

- fire, smoke and explosions of war films
- fast cars look furious
- starfighters in sci-fi epics scream

https://marvelofficial.com/product/wearable-iron-man-mk-7-suit-full-body-costume/

https://www.reddit.com/r/cyberpunkgame/comments/y2khrv/rtx_on_vs_off_in_your_opinion_is_raytracing_worth/

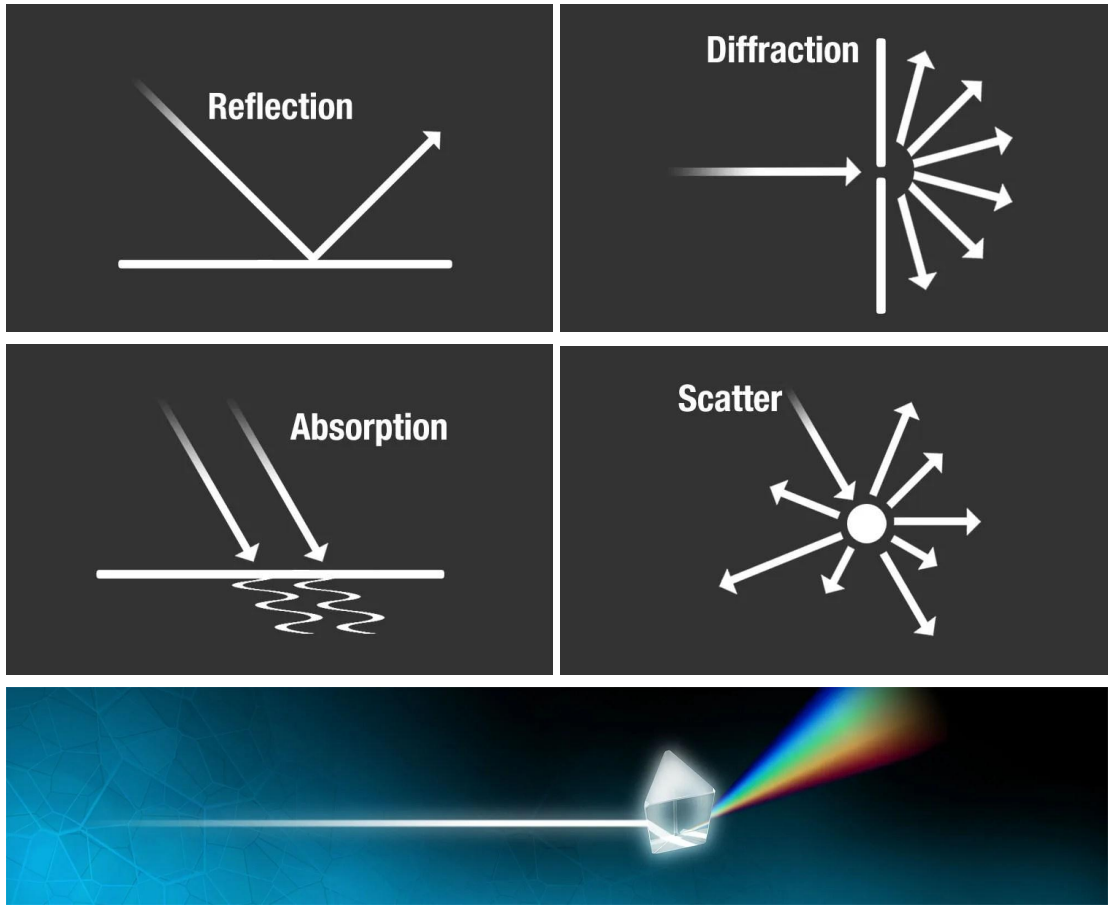
Definition

“ Ray tracing is a rendering technique that can realistically ***simulate the lighting of a scen****e and its objects by* ***rendering physically accurate reflections, refractions, shadows, and indirect lighting. “***

- March 19, 2018 Brian Caulfield: Nvidia

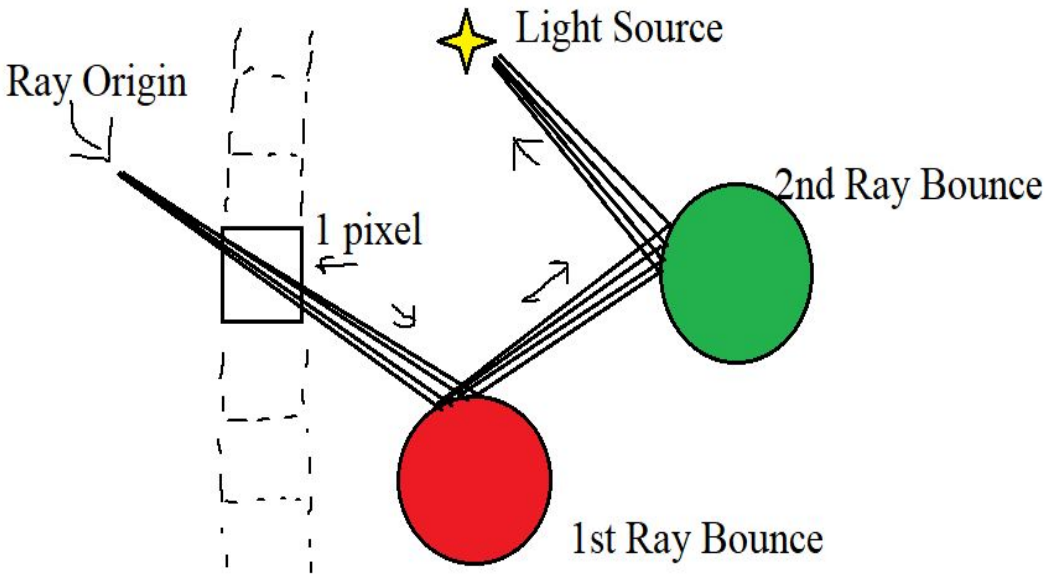
Problem Outline

- It is **computationally expensive** due to the **complexity of simulating light interactions with objects.**
- Light source emits **infinite number of rays**
- The primary challenge is the **high computational cost** associated with rendering images using ray tracing.
- Objective is to **reduce rendering time while maintaining image quality** through parallelization techniques.

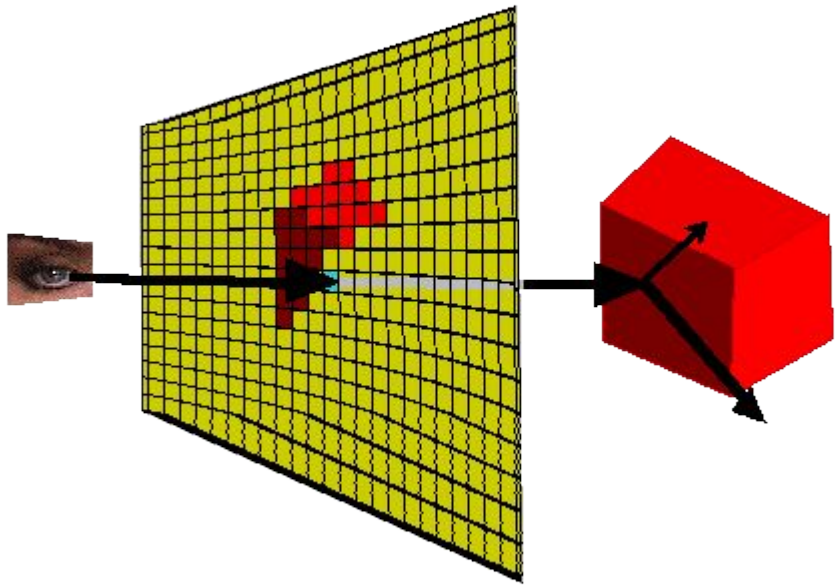


https://science.nasa.gov/ems/03_behaviors/

Observation



https://www.reddit.com/r/raytracing/comments/rv1er1/sample_per_pixel_and_ray_per_pixel_in_ray_and/

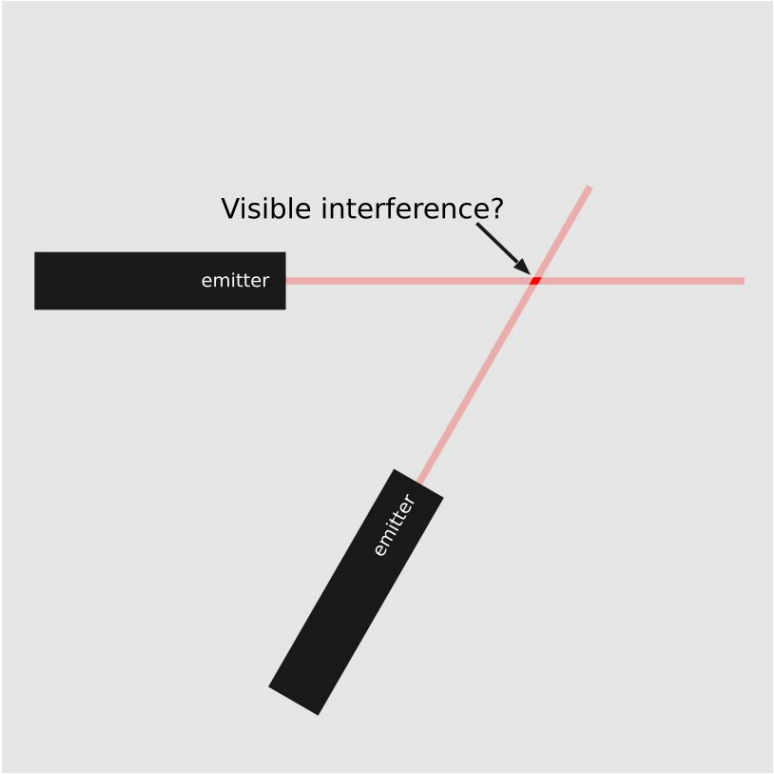


https://courses.washington.edu/arch481/1.Tapestry%20Reader/4.Rendering/8.Ray%20Tracing/0.default.html

Observation

- **Independence of task:** Each pixel independent of another pixel.
- **Data independence:** Same tracing computation need to be done on whole image (data).
- **Speed up potential**: Large enough problem size.
- **Resource optimisation:** Resource utilisation can be improved.

!! Parallelize !!



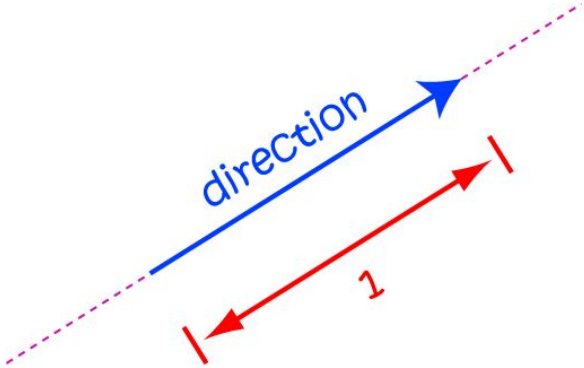
https://physics.stackexchange.com/questions/487918/can-light-beams-e-g-from-a-laser-visibly-interfere-if-they-cross-in-mid-air

Project setup

Prerequisites

Mathematical Basics:

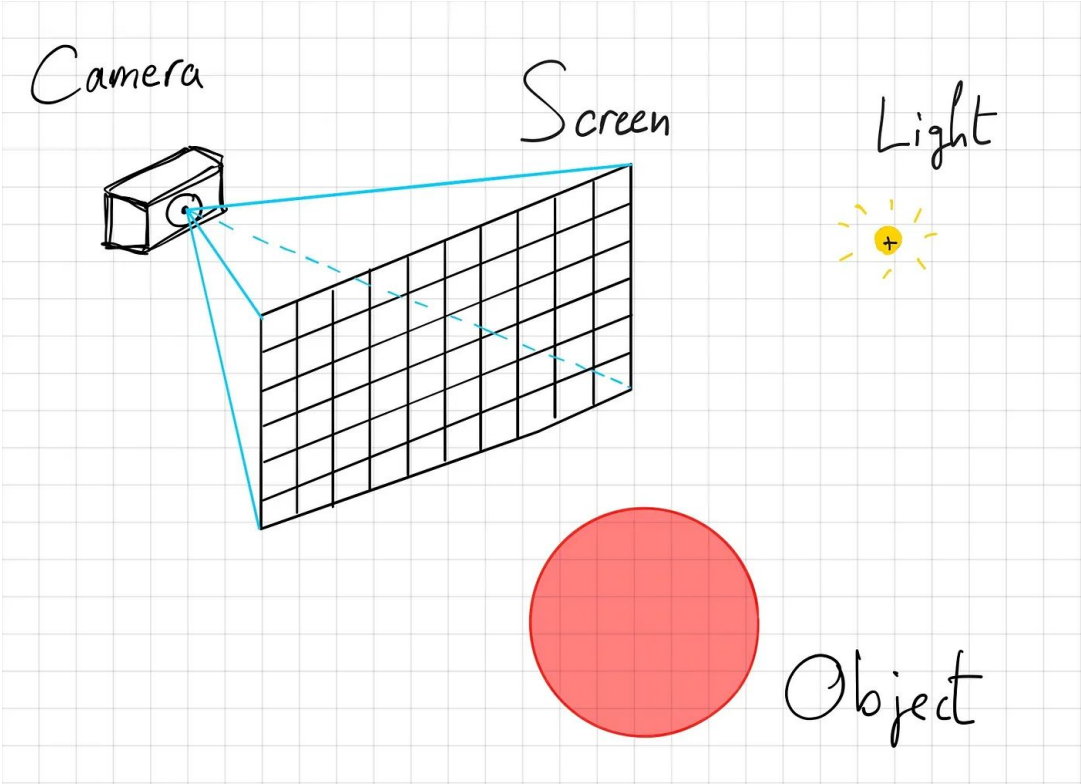
- Vector operations (addition, subtraction, length, normalization)
- Unit vector
- Direction of vector.
- Dot product and solving quadratic equations.



https://byjus.com/vector-formulas/

Components

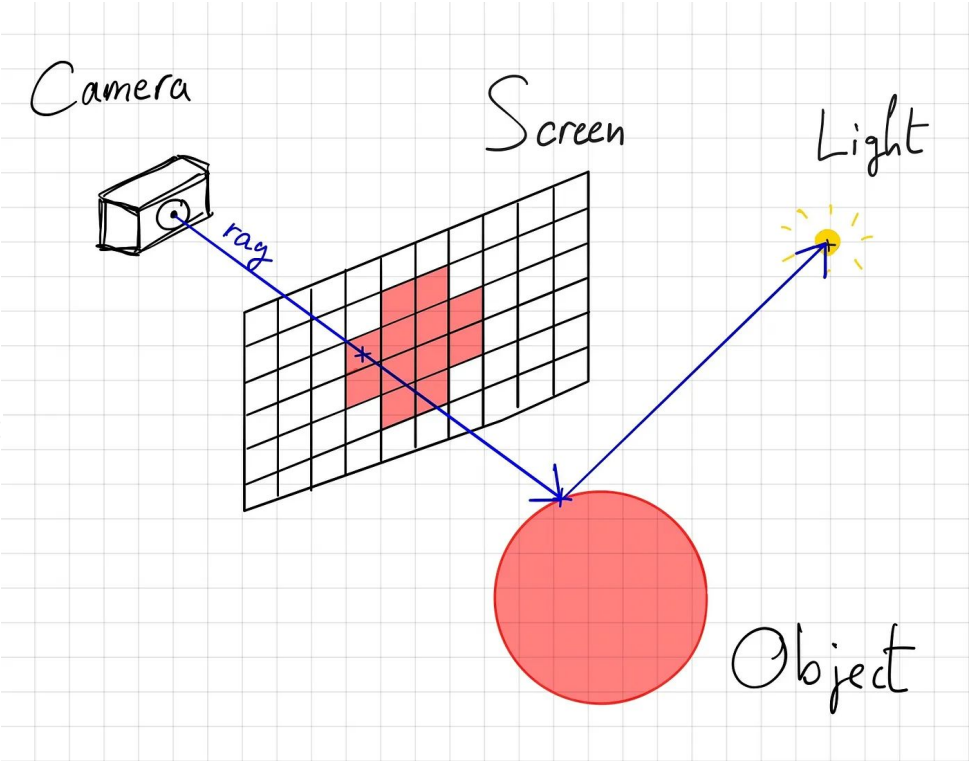
- **3D Space:** Objects positioned with three coordinates.
- **Objects:** For simplicity, consider spheres.
- **Light Source:** A point emitting light in all directions.
- **Camera:** A position from which the scene is observed.
- **Screen:** Rectangular plane through which the camera views the objects.



https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

Sudo Code Behind

```
for each pixel of the screen:
    pixel.color = black
    ray<direction>= generateRay(camera, pixel)
    if extended ray at pixel intersects any object of the scene then:
        calculate the intersection point (i) to the nearest object
        if intersection == true && no object between point of intersection (i) and Light (l):
            pixel.color =  calculateColor(object,i,l)
```

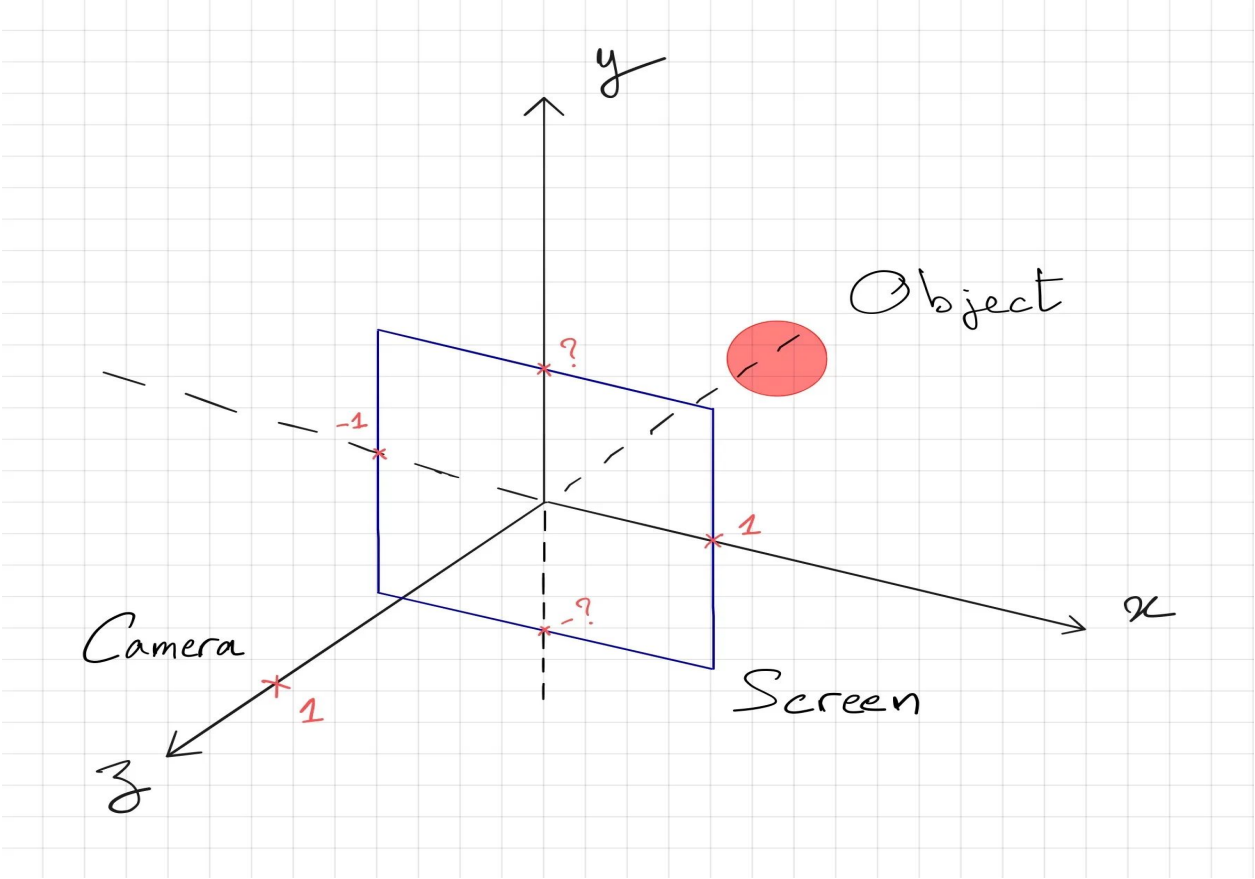


https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

```
for each pixel of the screen:
  pixel.color = black
  ray [direction]= generateRay(camera, pixel)
  if extended ray at pixel intersects any object of the scene then:
    calculate the intersection point (i) to the nearest object
    if intersection == true && no object between point of intersection (i) and Light (l):
      pixel.color =  calculateColor(object,i,l)
```

The Scene

- **camera** is located at the point **(x=0, y=0, z=1)**
- the **screen** is part of the plane formed by the **x** and **y** axes**.**



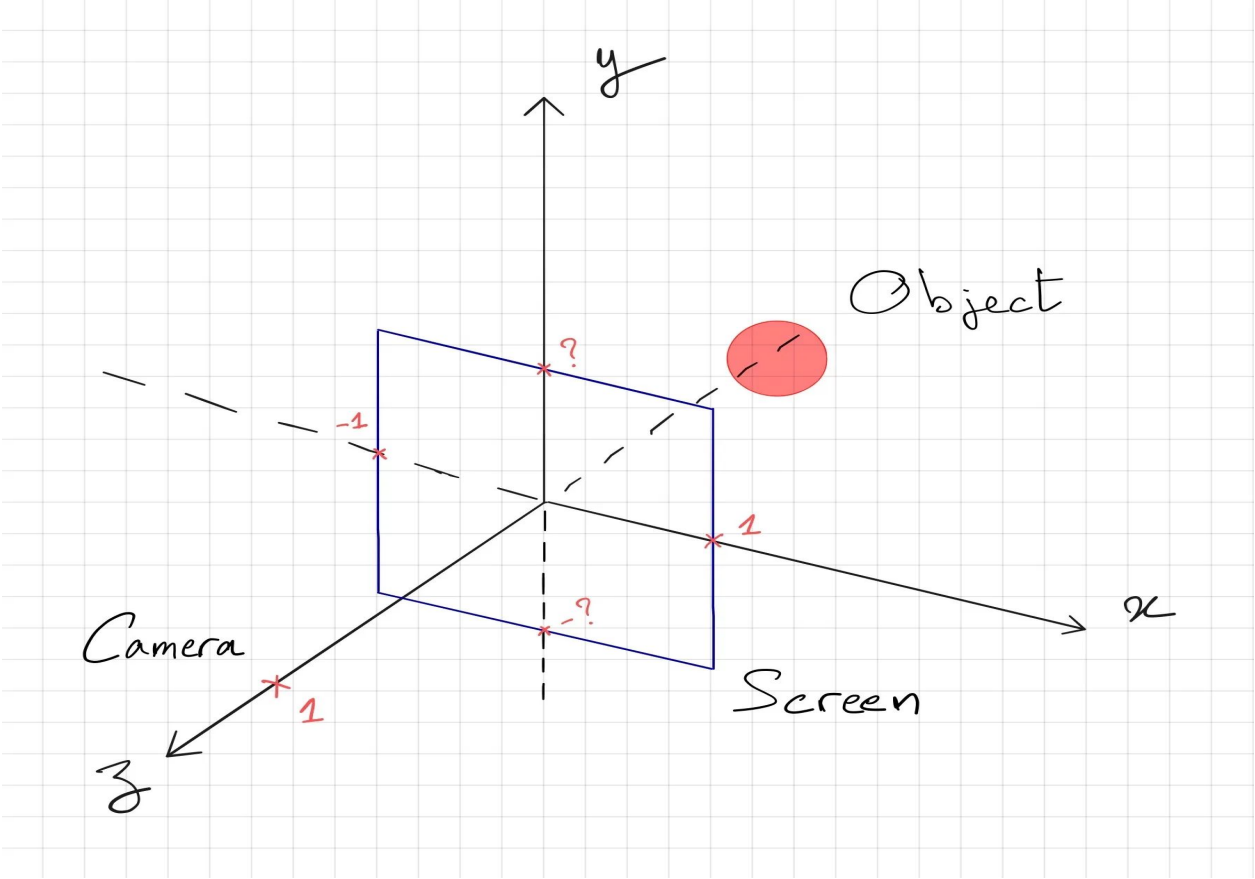
https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

Ray Intersection

```
for each pixel of the screen:
  pixel.color = black
  ray [direction]= generateRay(camera, pixel)
  if extended ray at pixel intersects any object of the scene then:
    calculate the intersection point (i) to the nearest object
    if intersection == true && no object between point of intersection (i) and Light (l):
      pixel.color =  calculateColor(object,i,l)
```

- **camera** is located at the point **(x=0, y=0, z=1)**
- the **screen** is part of the plane formed by the **x** and **y** axes**.**

$$ray(t) = camera + \frac{pixel - camera}{\|pixel - camera\|} t$$



https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

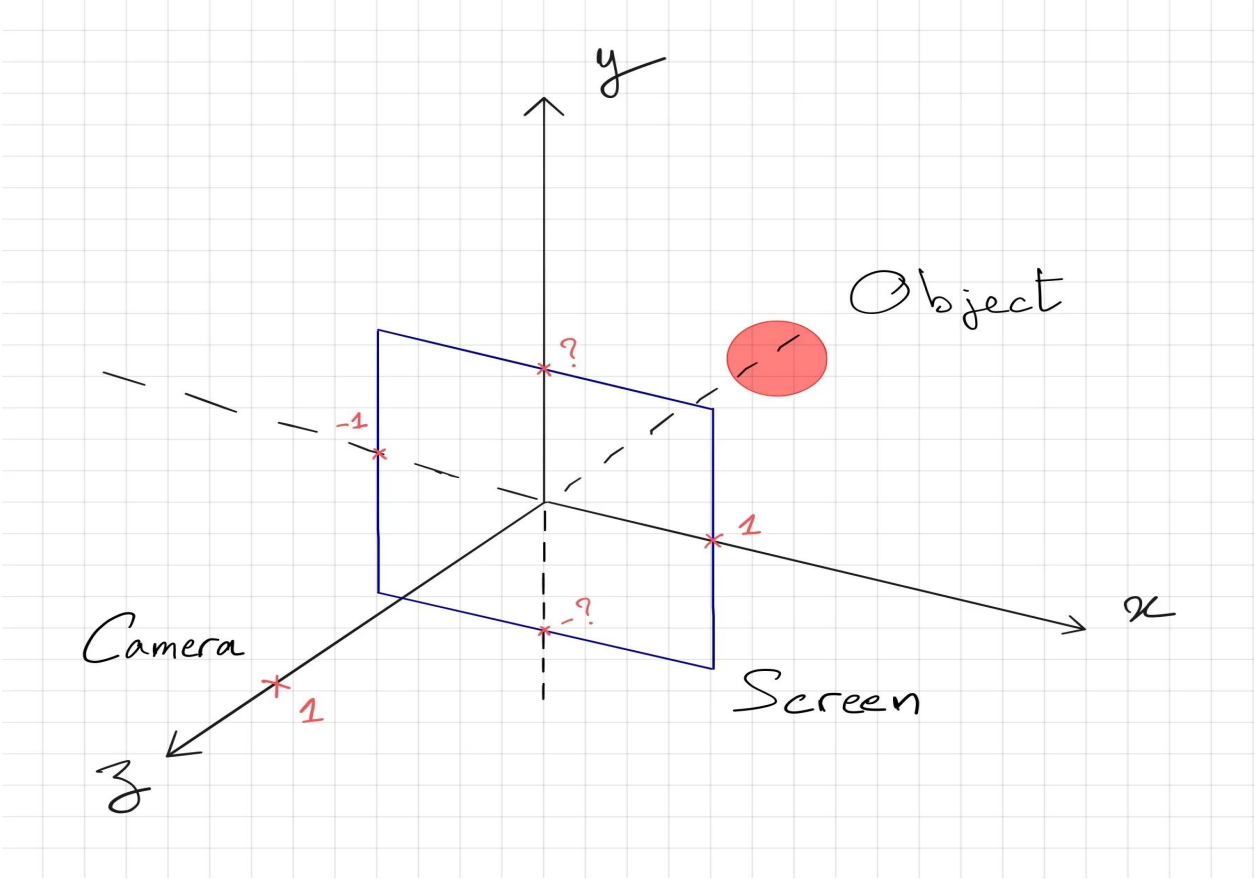
Ray Intersection

```
for each pixel of the screen:
  pixel.color = black
  ray [direction]= generateRay(camera, pixel)
  if extended ray at pixel intersects any object of the scene then:
    calculate the intersection point (i) to the nearest object
    if intersection == true && no object between point of intersection (i) and Light (l):
      pixel.color =  calculateColor(object,i,l)
```

- **camera** is located at the point **(x=0, y=0, z=1)**
- the **screen** is part of the plane formed by the **x** and **y** axes**.**

$$ray(t) = camera + \frac{pixel - camera}{\|pixel - camera\|} t$$

there is nothing special about ***camera*** or ***pixel***, we can similarly define a ray that starts at **origin (O)** and goes towards **destination (D)**



https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

Ray Intersection

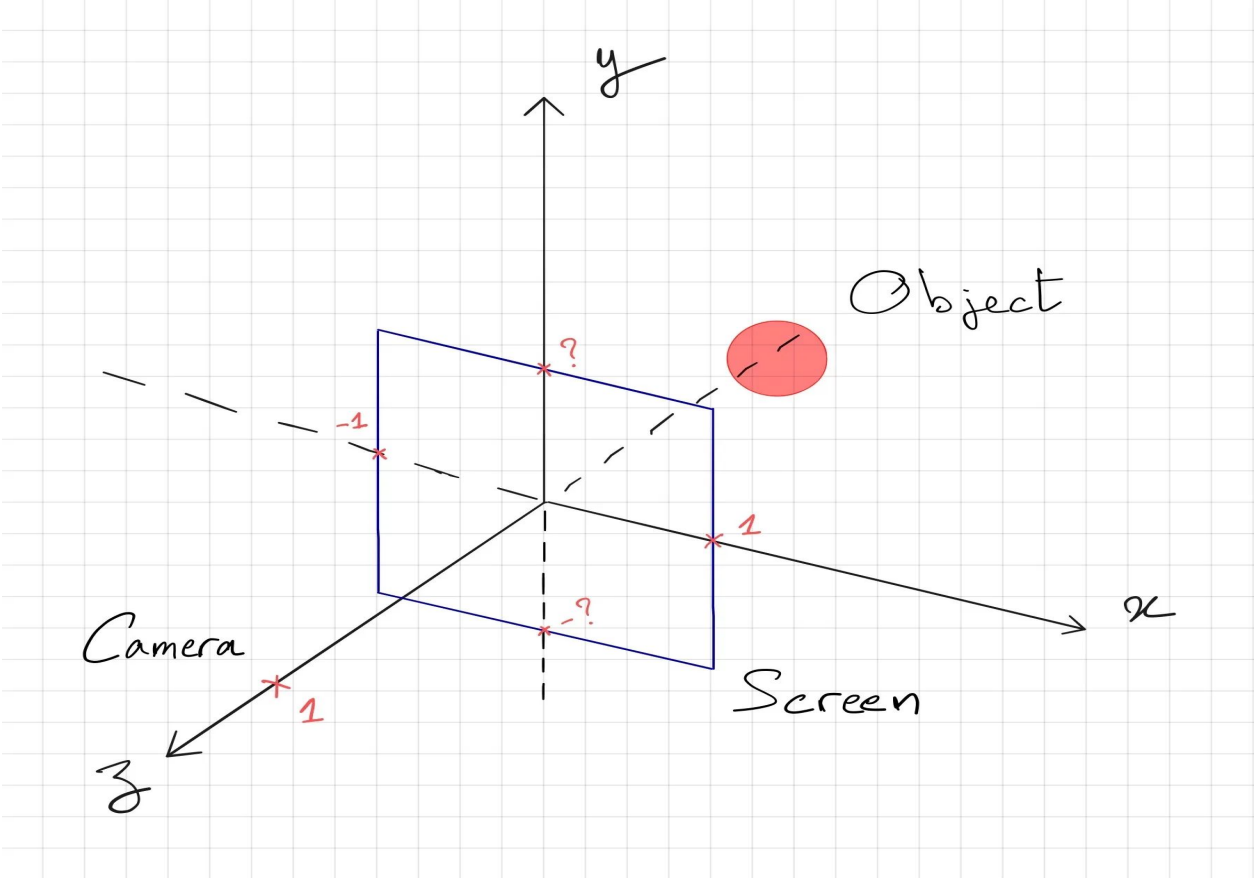
```
for each pixel of the screen:
  pixel.color = black
  ray [direction]= generateRay(camera, pixel)
  if extended ray at pixel intersects any object of the scene then:
    calculate the intersection point (i) to the nearest object
    if intersection == true && no object between point of intersection (i) and Light (l):
      pixel.color =  calculateColor(object,i,l)
```

- **camera** is located at the point **(x=0, y=0, z=1)**
- the **screen** is part of the plane formed by the **x** and **y** axes**.**

$$ray(t) = camera + \frac{pixel - camera}{\|pixel - camera\|}t$$

$$ray(t) = O + \frac{D - O}{\|D - O\|}t$$

$$= O + d \cdot t$$



https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

```
for each pixel of the screen:
  pixel.color = black
  ray [direction]= generateRay(camera, pixel)
  if extended ray at pixel intersects any object of the scene then:
    calculate the intersection point (i) to the nearest object
    if intersection == true && no object between point of intersection (i) and Light (l):
      pixel.color =  calculateColor(object,i,l)
```

The Object (sphere)

- **Simple Mathematical object**
- 2 important parameters: **Center, Radius.**
- **Ground** made up of a **giant sphere**
- **1 sphere** in **low complexity** execution
  3 spheres in **medium complexity** execution
  8 spheres in **high complexity** execution

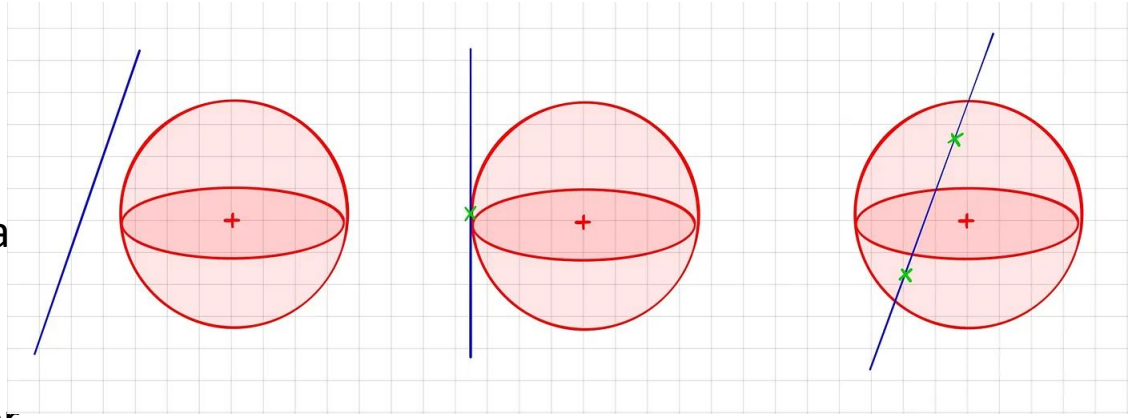
$$\|X - C\| = r$$

https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

```
for each pixel of the screen:
  pixel.color = black
  ray [direction]= generateRay(camera, pixel)
  if extended ray at pixel intersects any object of the scene then:
    calculate the intersection point (i) to the nearest object
    if intersection == true && no object between point of intersection (i) and Light (l):
      pixel.color =  calculateColor(object,i,l)
```

Sphere intersection

- **To consider:** only 3rd case
- Extended ray will show **t distance** from origin or ra to nearest point intersection
- Two Values of t, namely **t1** and **t2,** Only to consider, nearest point of intersection**, i.e. smaller value of t.**



$$\|X - C\| = r$$

$$\|ray(t) - C\|^2 = r^2$$

$$\begin{aligned} ray(t) &= O + \frac{D - O}{\|D - O\|} t \\ &= O + d \cdot t \end{aligned}$$

https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

So far

```
for each pixel of the screen:
    pixel.color = black
    ray [direction]= generateRay(camera, pixel)
    if extended ray at pixel intersects any object of the scene then:
        calculate the intersection point (i) to the nearest object
        if intersection == true && no object between point of intersection (i) and Light (l):
            pixel.color =  calculateColor(object,i,l)
```

no **object** between **point of intersection (i)** and **Light (l)**:

- **The light:** send infinite rays in all direction
- Point of intersection should deflect ray to light in order to be illuminated.
- To find if ray goes to light:
  Re use same task, send ray to all nearest object:
  If nearest object returned is closer than light: Obstruction
  else light

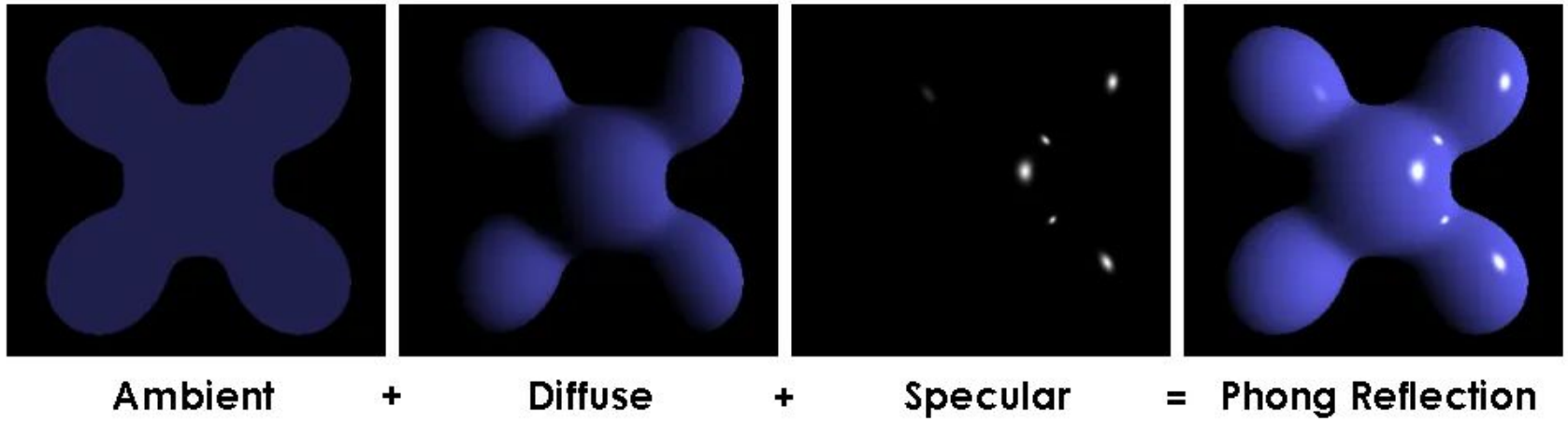
```
for each pixel of the screen:
    pixel.color = black
    ray [direction]= generateRay(camera, pixel)
    if extended ray at pixel intersects any object of the scene then:
        calculate the intersection point (i) to the nearest object
        if intersection == true && no object between point of intersection (i) and Light (l):
            pixel.color =  calculateColor(object,i,l)
```

Blinn-Phong reflection model

Blinn-Phong model: What does Camera see?

According to this model, any material has 4 properties:

- **Ambient color:** color that an object is suppose to have in absence of light.
- **Diffuse color:** color that is the closest to what we think of when we say “color”;
- **Specular color:** color of the shiny part of an object when light has stroke on it. Most of the time this is white;
- **Shininess:** a coefficient representing how shiny an object is;



https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

Blinn-Phong model: What does Camera see?

Illumination of a point is given by:

$$I_p = k_a * i_a + k_d * i_d * L \cdot N + k_s * i_s * (N \cdot \frac{L+V}{\|L+V\|})^{\frac{\alpha}{4}}$$

eq. 7 — Blinn-Phong model

- ka, kd, ks - coefficients of objects
- ia, id, is - coefficients of light
- L, N, V - unit vector
- α - shininess

Final Algorithm

Tracing the reflected ray

- Add Reflections: Compute reflected rays on intersections for realistic rendering.
- Max depth: Number of times to keep tracking the a reflected ray.

Color of pixel is given by:

$$c_p = i_0 + r_0 i_1 + r_0 r_1 i_2 + r_0 r_1 r_2 i_3 + \ldots$$

eq. 8 — color computation

https://omaraflak.medium.com/ray-tracing-from-scratch-in-python-41670e6a96f9

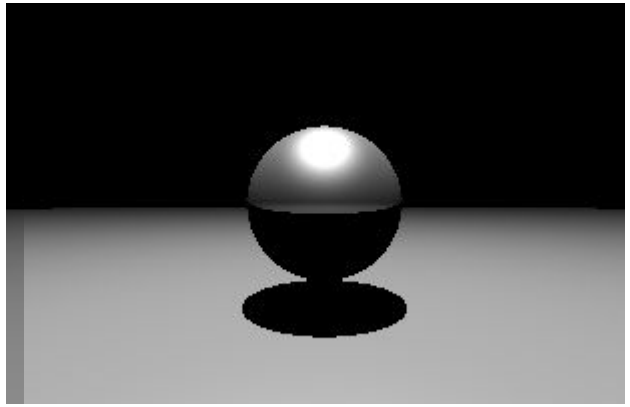
Experimentation and Result

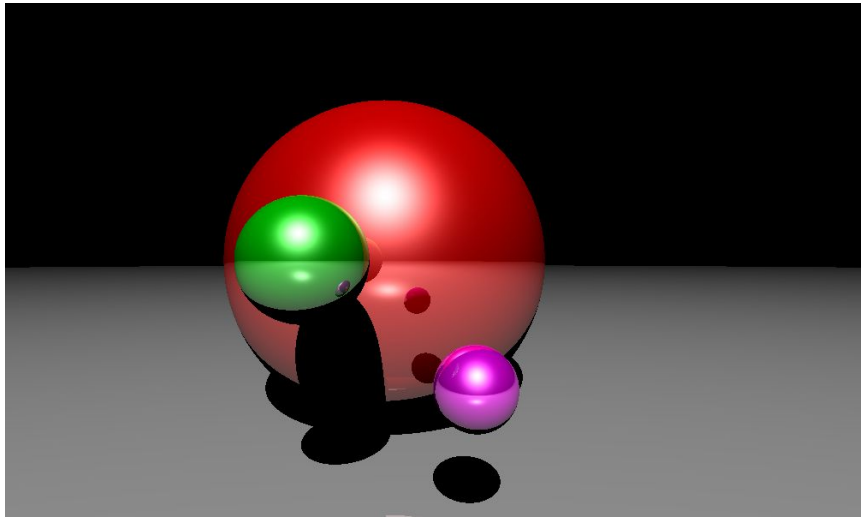
Experimentation Setup

Configuration

- Low complexity
  - Resolution - 300 x 200
  - Maximum depth - 5
  - Objects - 1

- Medium complexity
  - Resolution - 1000 x 600
  - Maximum depth - 30
  - Objects - 3





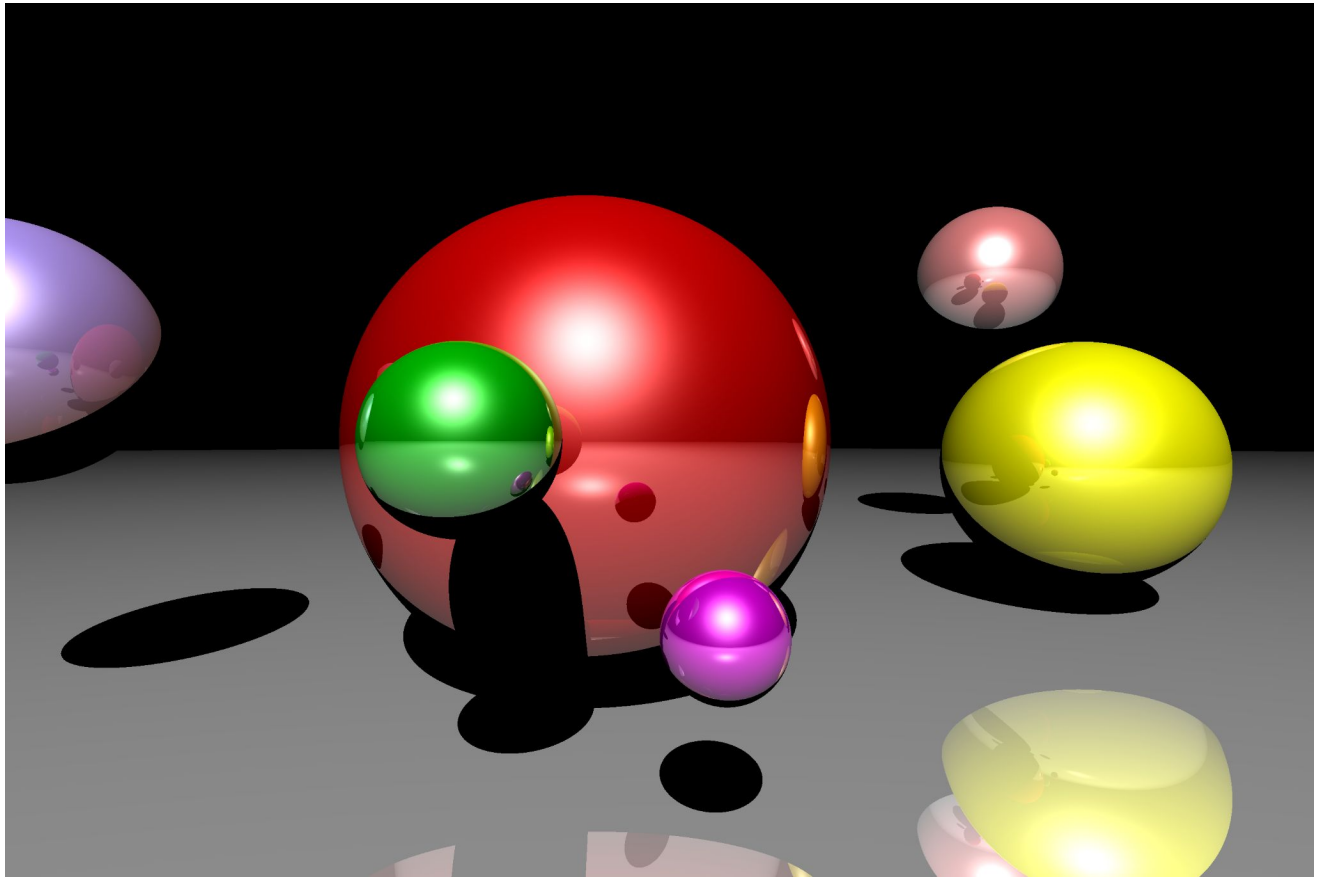
Experimentation Setup

Configuration

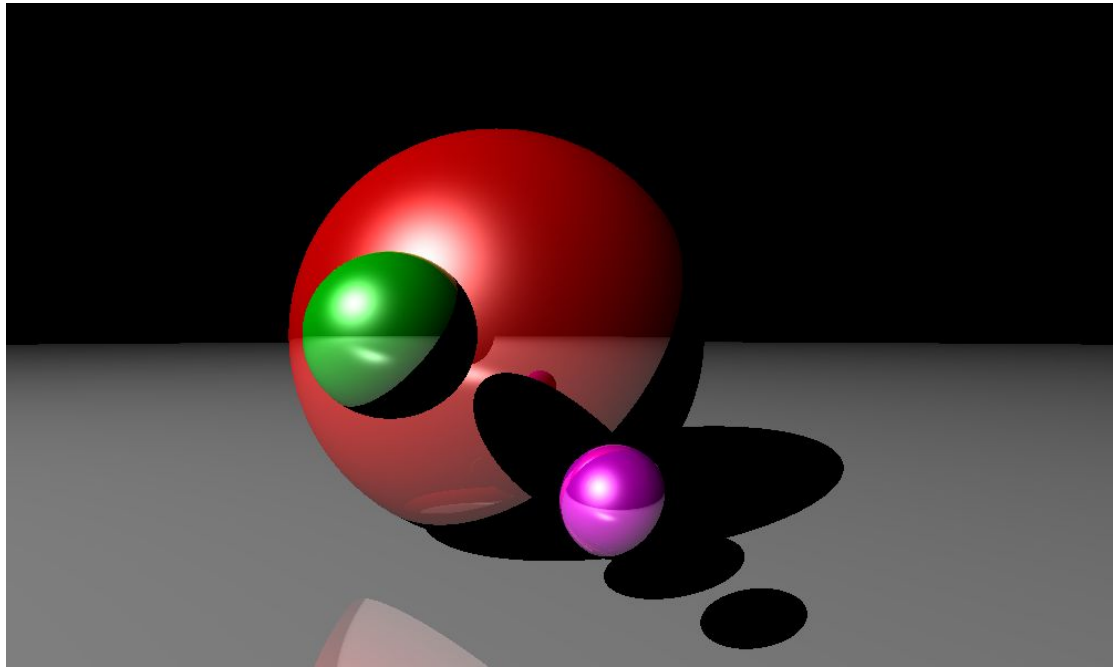
- High complexity
  - Resolution - 3000 x 2000
  - Maximum depth - 50
  - Objects - 6

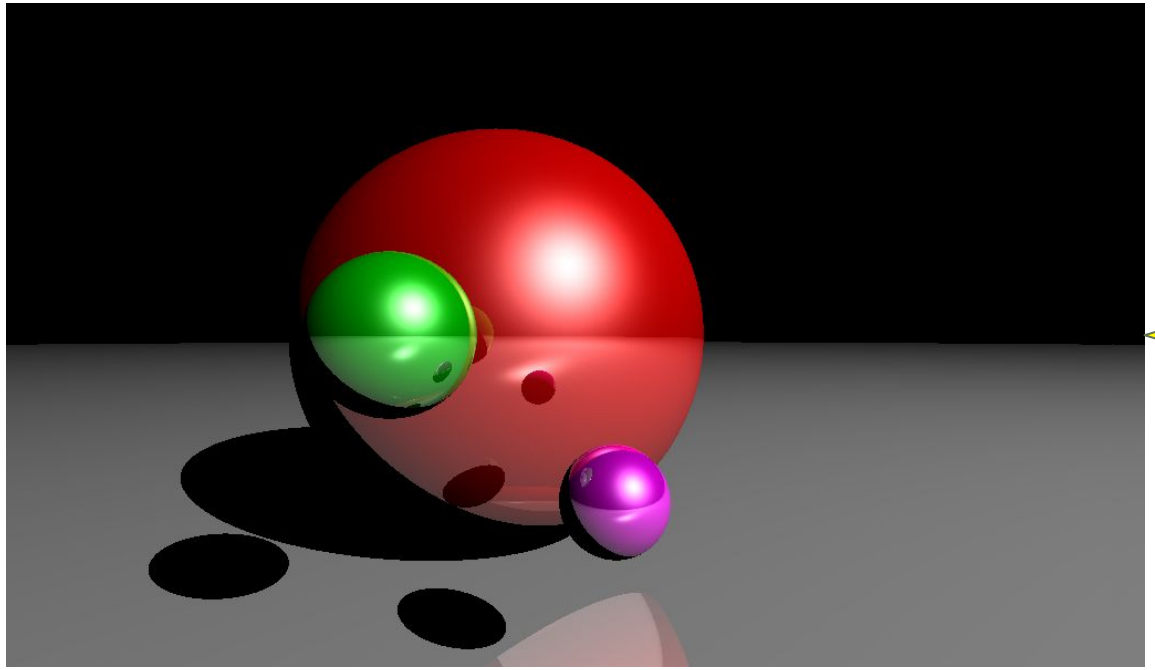
Parallel Processing

- MPI in Python
- Rows divided among processes
- Independent pixels

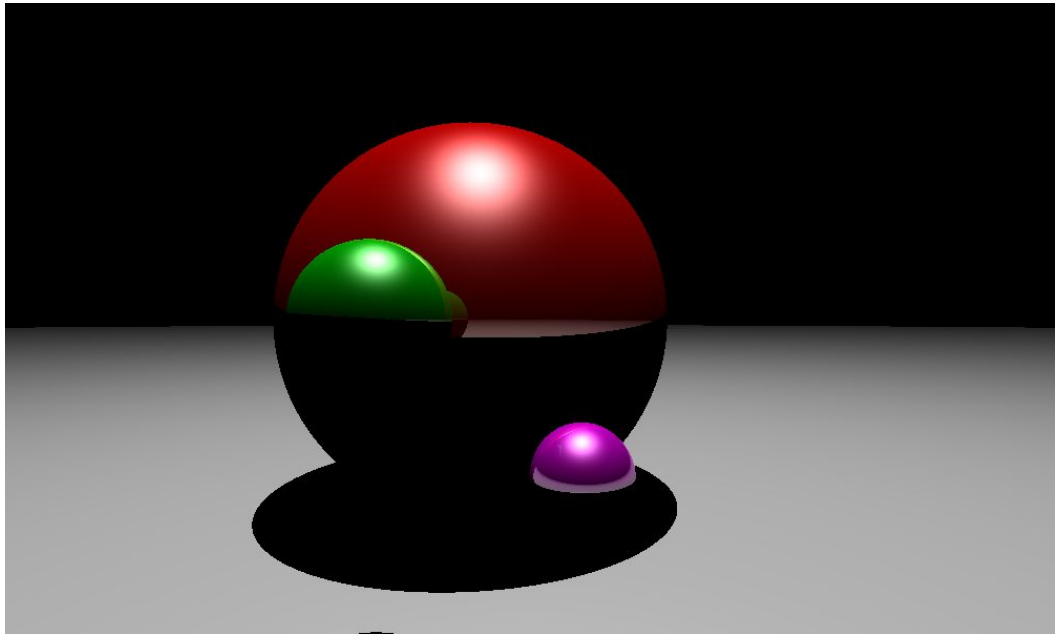


Rendered Image

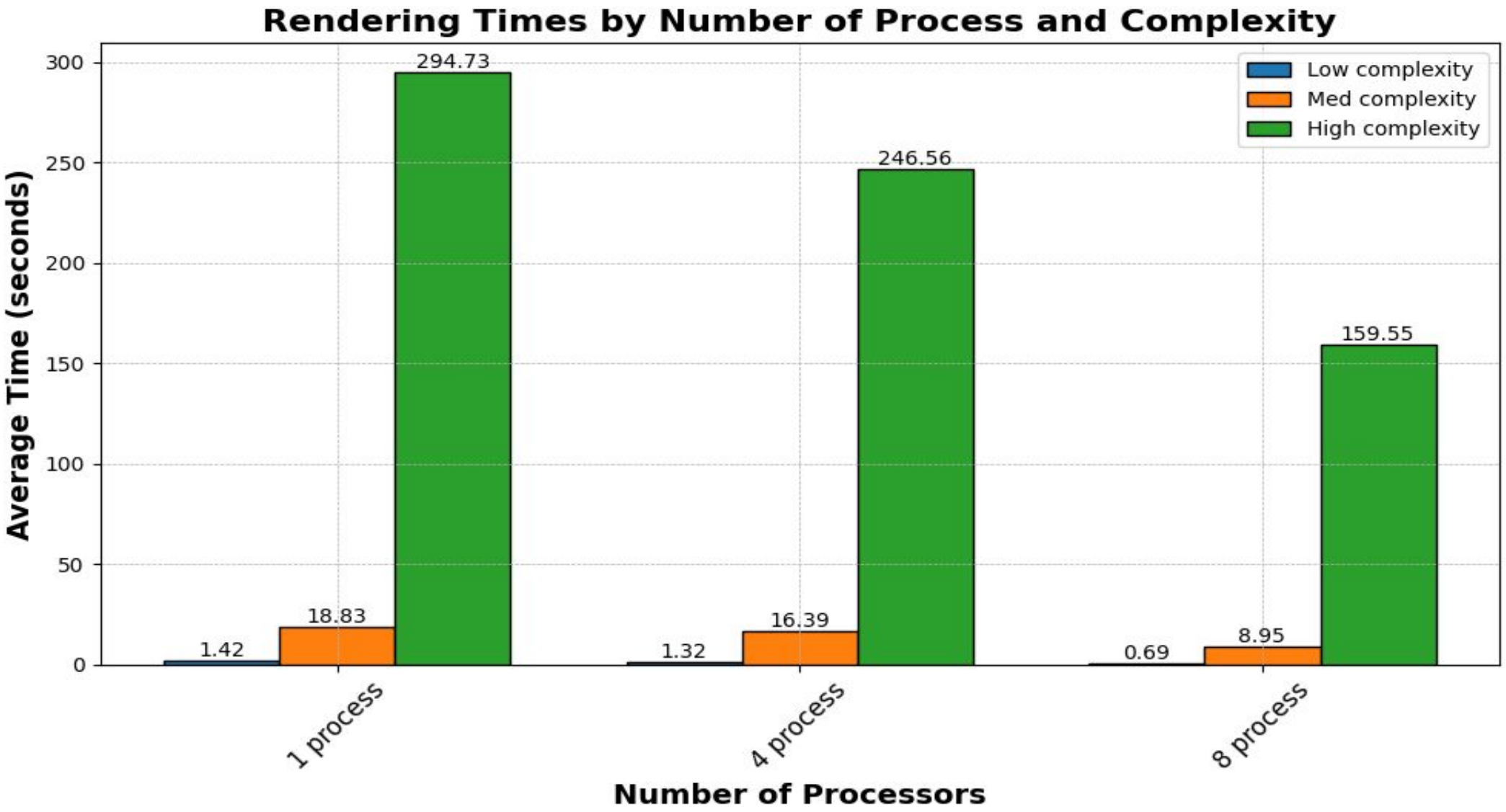




Rendered Image

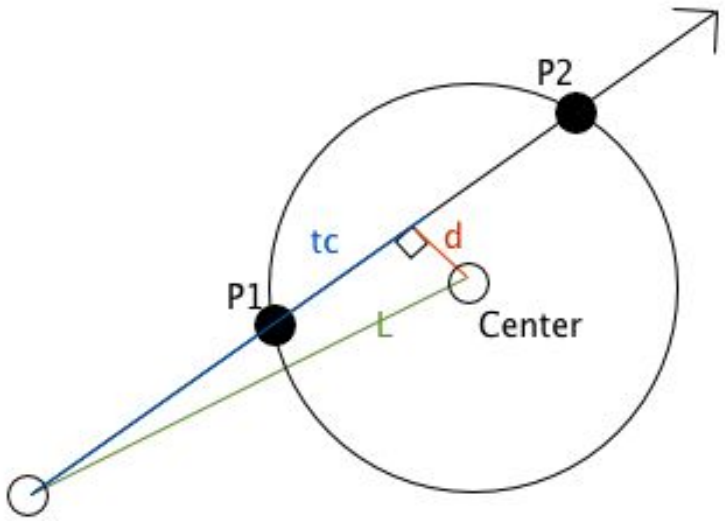


Comparison: Serial vs Parallel



Additional Experiment: Sphere Intersection using Geometric

- **Vector Calculation:** Calculate the vector from the ray origin to the sphere center
- **Closest Approach:** Project this vector onto the ray direction to find the closest point to the sphere center along the ray
- **Perpendicular Distance:** If this distance is greater than the radius squared, the ray misses the sphere.
- **Intersection Points:** Using the Pythagorean theorem, calculate the distance from the closest point to the intersection points.
- **Intersection Distance**: Calculate the actual distances along the ray to the intersection points:



30% reduction in Rendering time!

Conclusion

Conclusion

- **Ray Tracing Overview**: Simulates light paths to create photorealistic images, used in films, games, and design.
- **Challenges**: High computational cost due to complex light interactions.
- **Optimization**: Parallel processing reduces rendering time while maintaining quality.
- **Experiments**: Demonstrated scalability and efficiency improvements with different complexity levels.
- **Takeaway**: Parallel ray tracing enhances realistic image generation, crucial for advanced computing applications.

Reference

- Kadir, Syukriah & Khan, Tazrian. (2008). Parallel Ray Tracing using MPI and OpenMP.

- Vasiou, Elena & Shkurko, Konstantin & Mallett, Ian & Brunvand, Erik & Yuksel, Cem. (2018). A detailed study of ray tracing performance: render time and energy cost. The Visual Computer. 34. 10.1007/s00371-018-1532-8.

- https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model

Questions?

THANK YOU!

~ Pranay Bhatia