Assignment 2

Introduction to Classes: Ray-Sphere Intersection Check

Introduction: In Computer Graphics, 'Ray Tracing' is a popular method of displaying objects with their reflective and refractive properties. The following picture has been created by ray tracing technique by continuously checking imaginary ray-sphere intersections in 3D space.

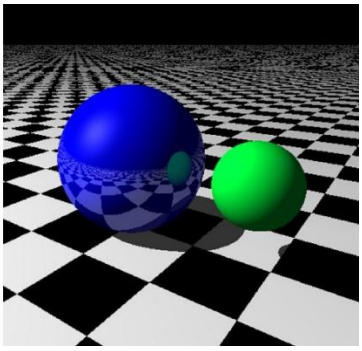


Figure 1: Objects displayed by ray tracing.

In this assignment, you will perform a simple check, whether a ray intersects a sphere. You need to implement three classes: **Point3D**, **Sphere**, and **Ray**.

Assignment Specification:

Point3D Class:

1) **Constructors for the Point3D class:**
   A point can be defined in 3D space by its x, y, and z components. The following expression defines a Point3D object **P** with its three member variables, **x**, **y**, and **z**. **x**, **y**, and **z** are of type **double**.

   Point3D P{x, y, z}; ……..(i)

   You need to implement the default constructor and constructor with parameters for Point3D class. Use C++11 uniform initialization. Take a look at the uploaded example on **Point2D**.

2) **Setters and getters**:
   Add setters and getters to set the values of member variables and access them respectively. This has been discussed in Week 1 lectures while object chaining.

3) **Add two Point3Ds**:
   Write a public method that will add a Point3D to an existing point ('this' Point3D) and then returns the reference to it ('this' Point3D). The function declaration will look like the following:

   Point3D &addPoints(const Point3D & P); ……(ii)

4) **Subtract one Point3D from another Point3D:**
Write a public method that will subtract a Point3D from an existing point ('this' Point3D) and then returns the reference to it ('this' Point3D). The function declaration will look like the following:

Point3D &subtractPoints(const Point3D & P); …… (iii)

5) **Multiply two Point3Ds:** Write a public method that multiplies **x**, **y**, and **z** member variables of a Point3D **P** with the existing Point3D's (this) **x**, **y**, and **z** member variables respectively. Next, add the values and return the sum.

double multiplyPoints(const Point3D& P) …… .(iv)

6) **Square of a Point3D**:This method will multiply x, y, and z member variables of this point with itself and return the sum.

double squarePoints() …… (v)

Sphere Class:

7) Constructors for the Sphere class:
A sphere can be defined by its center and radius. The center of a Sphere is a Point3D object; the radius is of type double. The Sphere class will include Point3D class. The following expression defines a Sphere object **S** with its center '**C**' and radius '**R**'.

Sphere S{Point3D C, double R}; ……..(vi)

You need to implement the default constructor and constructor with parameters for Sphere class. Use C++11 uniform initialization.

8) **Setters and getters**: Add setters and getters to set the values of member variables and access them respectively.

Ray Class:

9) Constructors for the Ray class:

A ray's origin is a Point3D object. A ray has its direction in the 3D space. Hence, both member variables of a ray can be defined as Point3D objects. A Ray object '**ray**' with its origin '**o**' and direction '**d**' has been defined below:

Ray ray{Point3D o, Point3D d };…….(vii)

You need to implement the default constructor and constructor with parameters for Ray class. Use C++11 uniform initialization.

10) **Setters and getters**: Add setters and getters to set the values of member variables and access them respectively.
11) **Friend class**: Declare Ray class as the friend of Sphere class.
12) **Check intersection with a sphere**: Write a member function for the Ray class that checks intersection with a sphere as follows:

    void checkIntersection(Sphere &s) …….. (ix)
    This function calculates the following: B^2 -AC……(x)

    where, for a ray with origin **o** and direction **d**, a sphere with radius **r** and center **c**, A, B, and C can be calculated as follows:

    A = $|\mathbf{d}^2|$, which can be calculated by **squarePoints()** when called by the direction member variable (i.e., d) of the ray.

    B = (o-c).d, which can be calculated as follows:
    - Step 1: Use **subtractPoints(Point3D &)** to subtract the center **c** from the origin **o**. This will return an updated **Point3D**;
    - Step 2: Use **multiplyPoints(const Point3D &)** to multiply the updated Point3D from Step 1 with the direction member variable **d** of the ray.

    and **C = $|\mathbf{o}-\mathbf{c}|^2 - \mathbf{r}^2$** ,which can be calculated as follows:

    - Step 1: Use subtractPoints(const Point3D&) to subtract **c** from **o**; this returns an updated Point3D;
    - Step 2: the updated Point3D from the Step 1 will call **squarePoints()**; this returns a double;
    - Step 3: Next, subtract the square of radius **r** from Step 2 to get the value of **C**.

Now, you can calculate the value of B^2 – AC. If this value is less than zero, the program will print the following:

> "Ray does not touch or intersect the sphere."
>
> Otherwise, it will print:
>
> "Ray either touches or intersects the sphere."

Use the attached **intersection_tester.cpp** file for testing purposes. You do not need to check invalid input in this assignment. Use the following cases for testing purpose:

Case 1:

Ray origin: (0, 2, 5)

Ray direction (1, 0, -2)

Sphere center (2, 4, 1)

Sphere radius 8

B*B – A*C = 300

The program will print: “Ray either touches or intersects the sphere.”

Case 2:

Ray origin: (0, 2, 5)

Ray direction: (1, 0, -2)

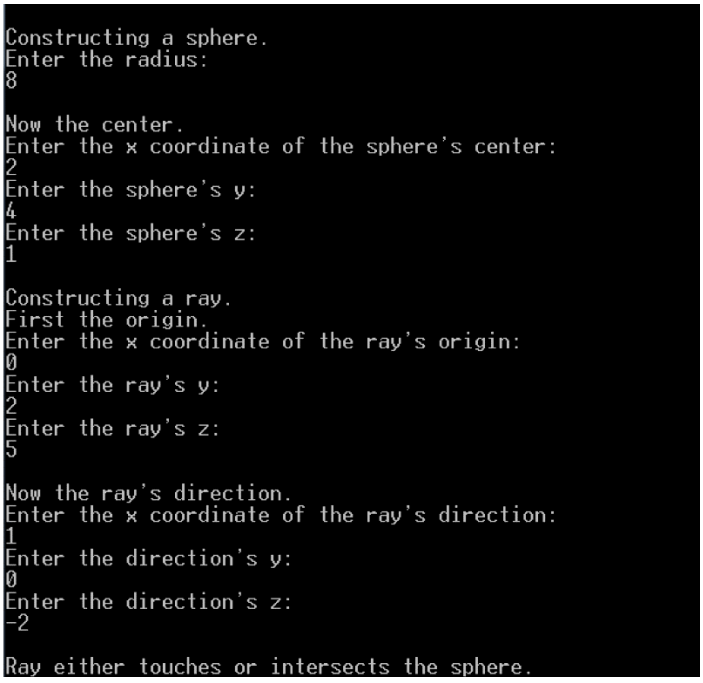
Sphere center: (10, -2, -5)

Sphere radius: 10

B*B – A*C = 320

The program will print: “Ray either touches or intersects the sphere.”

Below is a screenshot generated by the attached test file **intersection_tester.cpp**.



Submission:

Compile and execute your program with g++ or Microsoft compiler.

Place your solution in a zipped file named with your last name followed by the first initial of your first name followed by 2 (ex: YasminS2.zip) and submit the solution via canvas.

Include at the top of your solution (in comments) your name, what compiler(s) you used.

Thus, your zip will contain the following:

- Point3D.h, Point3D.cpp, Sphere.h, Sphere.cpp, Ray.h, and Ray.cpp

Submission deadline is Wednesday, April 17, 11:59 pm.

This assignment weighs 10% of the course.