

Assignment 2 - Kallem G Thomas 2304300

by Kallem Thomas

Submission date: 27-Dec-2024 10:41PM (UTC+0000)

Submission ID: 246905765

File name: 49116_Kallem_Thomas_Assignment_2_-_Kallem_G_Thomas_2304300_628793_921810234.docx
(584.63K)

Word count: 643

Character count: 3090

Games architecture Assignment 2

Submitted by: Kallem Thomas (Student ID: 2304300)

Module: Games Architecture and Mechanics (ACCB5017)

Lecturer: Nathan Morgan

Table of Contents

Distance between two points.....	1
Formulas used.....	1
Explanation	2
Implementation.....	2
A plane class	3
Formulas used.....	3
Explanation	4
Implementation.....	4
A Circle class	5
Formulas used.....	5
Explanation	5
Implementation.....	6
My code	7
References	8

Distance between two points

Formulas used

Euclidean distance Formula: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$

[1][2]

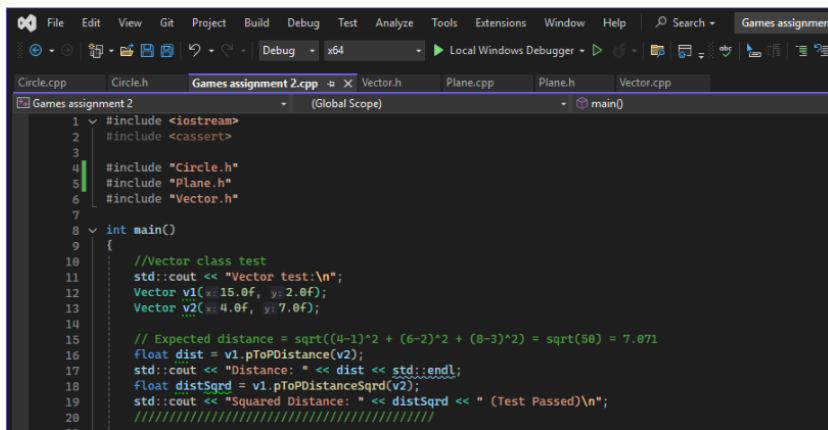
Explanation

This formula is a further example of the Pythagorean theorem ($a^2 + b^2 = c^2$).

As seen in the formula you could replace x2 and x1 for A, and then y1 and y2 for B and the formula would be the same. The only difference is that the X and Y are positions of a single point location, to find the difference between said two points is to have two X locations and find the difference by subtracting and doing the same to the Y locations of the two locations and then again for the z axis. By squaring the operation, this can give an absolute number to a negative value, for example $(-3)^2=9$ therefore regardless of whether X, Y or Z is negative will always result in a positive number[2].

Implementation

main



```
1 #include <iostream>
2 #include <cassert>
3
4 #include "Circle.h"
5 #include "Plane.h"
6 #include "Vector.h"
7
8 int main()
9 {
10     //Vector class test
11     std::cout << "Vector test:\n";
12     Vector v1(x: 15.0f, y: 2.0f);
13     Vector v2(x: 4.0f, y: 7.0f);
14
15     // Expected distance = sqrt((4-1)^2 + (6-2)^2 + (8-3)^2) = sqrt(50) = 7.071
16     float dist = v1.pToPDistance(v2);
17     std::cout << "Distance: " << dist << std::endl;
18     float distSqr = v1.pToPDistanceSqr(v2);
19     std::cout << "Squared Distance: " << distSqr << " (Test Passed)\n";
20     //////////////////////////////////////
21 }
```

Class

```

#pragma once
#include <cmath>
class Vector
{
public:
    float x, y, z;

    Vector(float x = 0, float y = 0, float z = 0) : x(x), y(y), z(z) {}

    float magnitude() const
    {
        return std::sqrt(x * x + y * y + z * z);
    }

    Vector normalize() const
    {
        float mag = magnitude();
        return Vector(x / mag, y / mag, z / mag);
    }

    float dot(const Vector& other) const
    {
        return x * other.x + y * other.y + z * other.z;
    }

    Vector operator-(const Vector& other) const
    {
        return Vector(x - other.x, y - other.y, z - other.z);
    }

    float pToPDistanceSqrd(const Vector& other) const
    {
        float dx = x - other.x;
        float dy = y - other.y;
        float dz = z - other.z;
        return dx * dx + dy * dy + dz * dz;
    }

    float pToPDistance(const Vector& other) const
    {
        return std::sqrt(pToPDistanceSqrd(other));
    }
};

```

Output

```

Vector:
Distance: 12.083
Squared Distance: 146

```

A plane class

Formulas used

$1e-6$

$Ax + By + Cz + d = 0$

Explanation

$1e-6$ is a formula used within mathematics and coding to get the floating-point error. The $1e-6$ is 1×10^{-6} which would be considered as 0.000001. Therefore, when used to detect whether the distance between the plane and the point is $1e-6$ smaller than it, it would detect even the slightest difference between the point and plane detection.

$Ax + By + Cz + d = 0$ is an equation where at least a, b or c must be a non-zero number. A plane is determined by a point and a Vector.[3]

Implementation

Class

```
class Plane
{
private:
public:
    Vector normal;
    float d;
    Plane( Vector normal, const Vector& point) : normal(normal.normalize())
    {
        d = this->normal.dot(point);
    }

    float distanceFromPoint(const Vector& p) const
    {
        return normal.dot(p) - d;
    }

    // Determine whether a point is on, in front of, or behind the plane
    std::string pointLocation(const Vector& p) const
    {
        float distance = distanceFromPoint(p);
        std::cout << "distance from the plane: " << distance << std::endl;
        if (std::abs(distance) < 1e-1)
        {
            return "On the plane\n";
        }
        else if (distance > 0)
        {
            return "In front of the plane\n";
        }
        else
        {
            return "Behind the plane\n";
        }
    }
}
```

Main

```

//Plane test
std::cout << "\nPlane Test:\n";
Vector normal(x:1, y:2, z:2);
Vector point(x:0, y:2, z:0);
Plane plane(normal, point); //2 different vectors

Vector testPoint1(x:1, y:0, z:0);
Vector testPoint2(x:0, y:1, z:0);
Vector testPoint3(x:1, y:0, z:10); //the X axis determines whether the point is in front or behind the plane
Vector testPoint4(x:0, y:2, z:555555);

std::cout << "Test Point 1: " << plane.pointLocation(testPoint1) << "\n"; //
std::cout << "Test Point 2: " << plane.pointLocation(testPoint2) << "\n";
std::cout << "Test Point 3: " << plane.pointLocation(testPoint3) << "\n";
std::cout << "Test Point 4: " << plane.pointLocation(testPoint4) << "\n";

```

Output

```

Plane Test:
distance from the plane: -1
Test Point 1: Behind the plane

distance from the plane: -0.666667
Test Point 2: Behind the plane

distance from the plane: 5.66667
Test Point 3: In front of the plane

distance from the plane: 370370
Test Point 4: In front of the plane

```

A Circle class

Formulas used

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

$$d > r_1 + r_2$$

Explanation

Each sphere must have a centre point and for this to exist the code already implemented before for the vector class is used to determine the co-ordinates of the centre point of the circle, as the code does not require area then the code does not

need an implementation of $A = \pi r^2$, however, Radius is still a necessary value to the code as to find the edge of the sphere there must be a distance from the centre of the circle. By using an example, a vector of centre is created at co-ordinates (0,0,0) with a radius of 2. By doing this the implementation now knows that the furthest it wants the circle to be is 2 points away. So now if there is another sphere at (4,0,0) with a radius of 2 again, that means that from the centre of the circle to now 2 points away in every direction from the centre of the sphere is created and will be checked for a collision, in this case there will be a collision at (2,0,0) exactly. On a sphere, any point on the surface is at a steady distance from the centre of the sphere, which is called the radius, this means that no matter at what point you select on a sphere, it would always equal to another points radius, however in this implementation there is no need for volume, this means that there is no need for $V = 4/3 \pi r^3$.

Implementation

Class

```

1  #pragma once
2  #include <iostream>
3
4  #include "Plane.h"
5  #include "Vector.h"
6
7  class Circle
8  {
9  public:
10     Vector center;
11     float radius;
12
13     Circle(const Vector& center, float radius) : center(center), radius(radius) {}
14
15     bool isColliding(const Circle& other) const
16     {
17         float distance = center.pToPDis((parameter) const Circle& other
18         return distance <= (radius + other.radius);
19     }
20
21     void display() const {
22         std::cout << "Circle Center: (x: " << center.x << ", y: " << center.y << ", z: " << center.z << ")"
23     }
24
25     bool testCircleCollision(const Plane& plane) const
26     {
27         float distance = plane.distanceFromPoint(center);
28         return distance <= radius;
29     }

```

Main

```
//Circle
Circle circle1(center: Vector(x:19, y: 20, z:15), radius:5);
Circle circle2(center: Vector(x:12, y: 17, z:15), radius:3); //will collide with circle 1
std::cout << "\nCircle Test:\n";
std::cout << "Circle 1:\n";
circle1.display();
std::cout << "Circle 2:\n";
circle2.display();

if (circle1.isColliding(circle2)) //detecting the collision between circles, however doesn't check collisions for the plane
{
    std::cout << "The circles are colliding.\n";
}
else
{
    std::cout << "The circles are not colliding.\n";
}
//=====
```

```
//Collision for circle to plane and point to plane
std::cout << "\nPoint to plane and sphere to plane collision:\n";
std::cout << "Circle 3:\n";

Plane plane2(Vector(0, 300, 20), 0); // Horizontal plane at y = 300
Vector point2(-2, 300, 2); //vector of the new point
Circle circle3(Vector(2, 900, 900), 1);
circle3.display();

std::string pointResult = plane2.pointLocation(point2);
bool circleCollision = circle3.testCircleCollision(plane2);

std::cout << "Point is " << pointResult << " relative to the plane." << std::endl;
std::cout << "Circle collision: " << (circleCollision ? "Yes" : "No") << std::endl;
return 0;

//Only god knows how this code works
//I asked santa for my code to work, and he did not deliver
//thank you santa...
```

```
//note: The filename for circle is circle because i had done a 2D version first to make it easier on myself but then edited it to be 3d
//which only meant adding in a new axis to the Vector.h class and then changing some of the objects sent to the vector. example: (2,0) => (2,0,0)
```

Output

```
Circle Test:
Circle 1:
Circle Center: (x: 19, y: 20, z: 15)
Radius: 5
Circle 2:
Circle Center: (x: 12, y: 17, z: 15)
Radius: 3
The circles are colliding.

Point to plane and sphere to plane collision:
Circle 3:
Circle Center: (x: 2, y: 900, z: 900)
Radius: 1
distance from the plane: 299.469
Point is In front of the plane
relative to the plane.
Circle collision: No
```

Code

[Download the Code by ctrl + click here](#)

References

References

- [1] "Distance Between Two Points Formula - Definition & Examples."
<https://byjus.com/maths/distance-between-two-points-formula/> (accessed Dec 23).
- [2] "Distance Between Two Points - Formula, Derivation, Examples."
<https://www.cuemath.com/geometry/distance-between-two-points/>
(accessed Dec 23).
- [3] "3D Coordinate Geometry - Equation of a Plane | Brilliant Math & Science Wiki." <https://brilliant.org/wiki/3d-coordinate-geometry-equation-of-a-plane/> (accessed Dec 27).

Assignment 2 - Kallem G Thomas 2304300

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

75/100

Great work Kallem

A decent engagement with literature

Well written report with a good understanding and implementation.

PAGE 1



Comment 1

Labelling an equation as a figure is typically done to maintain consistency with the formatting style used in the document. In some academic styles, such as in certain fields of science and engineering, equations are treated as visual elements or figures, especially when they are central to the content of the paper. By labelling an equation as a figure, it allows for clear referencing throughout the text and ensures that it is easily distinguishable from the main narrative. This approach helps readers locate and focus on key mathematical expressions more effectively.

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8