

Steffan Wilkins Games architecture Report

by Steffan Wilkins

Submission date: 09-Jan-2023 10:53PM (UTC+0000)

Submission ID: 194259623

File name: 32671_Steffan_Wilkins_Steffan_Wilkins_Games_architecture_Report_473919_82843107.odt
(1.07M)

Word count: 1178

Character count: 5321

BSc Computer Games Development

Games Architecture and Mechanics

ACCB5017

Assignment 2

Steffan Wilkins - 2101242

Table of Contents

Introduction 3

Functions..... 4

Vector Class..... 7

Plane Class 7

Sphere Class 8

Test Program..... 9

Results..... 10

Bibliography 11

Introduction

In this report I have commented on my code and given images as proof that it works for my assignment. In this assignment it was split into four parts where in part one I had to create a program to calculate the distance of one point to another, then calculate the distance but in squared space. In part two I had to make a function to calculate the distance between a sphere and a point with just a vector a point on the line and a random point. Then in part three I had to calculate the distance between one sphere and another. Then finally in part four I had to calculate the distance between a plane and sphere. In this report you will see a section when I describe every function I've made with its inputs and outputs and if I believe that my function is correct or not. Then there are images of my main class that has all the inputs and the executions of all the functions. Finally, there are images included to show the classes needed such as planes, spheres, and vectors, then there is the command prompt with all my answers that is given in my code.

The only source of information I used for this assignment was the resources provided to me on Moodle by my tutor. [1]

Functions

```
float Vector::pToPDistance(Vector& v1) //This calculates the distance from one point to another
{
    float Dist = sqrt(pow(x - v1.x, 2) + pow(y - v1.y, 2) + pow(z - v1.z, 2));

    return Dist;
}

float Vector::pToPDistanceSqrd(Vector& v1) //This calculates the distance from one point to another in squared space
{
    return pow(x - v1.x, 2) + pow(y - v1.y, 2) + pow(z - v1.z, 2);
}
```

The *pToPDistance* function uses the x, y and z coordinates of two points and then uses the point-to-point formula to calculate the distance between each point. In my test I used the values 1, 2 and 3, for one point then the other point had a value of 2, 3 and 4. When my program is run and the calculation is run it gives a result of 1.73205 when I check with a second-party calculator it gave the same result, showing that this program is correct.

The *pToPDistanceSqrd* function works similarly to the previous function however whereas, the previous function worked in real space the *pToPDistanceSqrd* function works in squared space which is more commonly used in games. As for my results with the same input, my result was 3 which is a round-up from 2.9999. This aligned with the expected answer therefore this function is a success.

```
void Plane::Normal(const Plane& v1) { //This gives the normal of the line with the vector of the line
    x = 1 / v1.x;
    y = 1 / v1.y;
    z = 1 / v1.z;
}

float Plane::LtoPDistance(Plane& p0, Plane& p1) { //This calculates the distance of the plane from the point

    Plane w(p1.x - p0.x, p1.y - p0.y, p1.z - p0.z);
    Plane vw(w.x - x, w.y - y, w.z - z);

    return sqrt((w.x * vw.x) + (w.y * vw.y) + (w.z * vw.z));
}

float Plane::CollisionDetect(Plane& p0, Plane& p1) { //This line locates the point in position to the plane

    float D = -1 * ((x * p0.x) + (y * p0.y) + (z * p0.z));
    float tx = x * p1.x, ty = y * p1.y, tz = z * p1.z;

    return (tx + ty + tz + D);
}
```

The *Normal* function takes an input of a vector and then normalises it by inverting the number. With inputs 4, 5 and 6, the outputs given are 0.25, 0.2 and 1.17. In this function, I am unsure of the equation to get a normal of a vector, however, the result given are appropriate therefore I would give this function a success.

LtoPDistance is a function that uses the normal of a line, a point on the line and a separate point to calculate the distance between the line and the point. For my inputs the normal is the result of the previous function, the point on the line has an input of 3, 4 and 5. Finally, the point off the line has a value of 5, 6 and 7. The expected result of the equation is to be not on the line but not too far away therefore with a result of 3.28 the function gave an appropriate answer therefore the function is correct.

CollisionDetect is the next function. Its purpose is to find the line regarding the plane. The results that are given could either be a positive number which means it's above the plane, 0 which means that the point is on the plane, or a negative number which means that the point is below the plane which means there's a collision. The input of this where the same as the previous function, and the result of this function was 1.23 which is what was

expected as it's a small number and positive then this function seems to be working correctly.

```
void Plane::CD(int location) { //this gives the result of if there is a collision between the plane and and the point
    if (location > 0) {
        cout << "No collision" << endl;
    }
    else if (location == 0) {
        cout << "Point on line" << endl;
    }
    else if (location < 0) {
        cout << "collision detected" << endl;
    }
}

void Plane::NSurface(Plane& p1, Plane& p2) { //This calculates the normal of the plane from two point on the plane
    Plane A(p1.x - x, p1.y - y, p1.z - z);
    Plane B(p2.x - x, p2.y - y, p2.z - z);

    x = A.y * B.z - A.z * B.y;
    y = A.z * B.x - A.x * B.z;
    z = A.x * B.y - A.y * B.x;
}

float Plane::StoPDistance(Plane& p0, Plane& s1) { //this calculates the distance between the plane and a spere
    Plane w(s1.x - p0.x, s1.y - p0.y, s1.z - p0.z);
    Plane ww(w.x - x, w.y - y, w.z - z);

    return sqrt((w.x * ww.x) + (w.y * ww.y) + (w.z * ww.z));
}
```

The *CD* function takes the result of the *CollisionDetect* function to output whether the output of the function means that there is no collision, the point is on the line, or there has been a collision. The expected output of this function should be "No collision", which when ran gave the expected results. Therefore, the function is a success.

NSurface is supposed to calculate the normal of a plane given two points in the plane. For this, I used the same position on the plane as with other functions previously, but I used the *CollisionDetect* to find another point on the plane, when I then put those numbers into the function the expected results should be the same normal as I've calculated previously. However, the result I got was -36, 57, and -24. These are absurd numbers therefore this function is unsuccessful.

The *StoSDistance* function calculates the distance from a sphere to a plane. I do this by inputting the centre point of the plain then calculating the distance from that point to the plane. Then it subtracts the radius of the sphere from the distance then outputs the answer. The expected result of this should be a positive number. With the sphere point of 7, 8, and 9, the point on the plane is, 2, 3, and 4, the feature vector being 3, 4, and 5 then being normalized, and a radius of 5. The answer came out to be 8.66, which is an expected result therefore this calculation is also correct.

```

void Plane::SPCollisionDetection(Plane& p0, Plane& s0, int s0r, float dist) { //This calculates if there is a collision or not between a plane and a sphere
    if (dist > s0r) {
        cout << "No collision detected" << endl;
    }
    else if (dist == s0r) {
        cout << "Objects are adjoint" << endl;
    }
    else if (dist < s0r) {
        cout << "Collision detected" << endl;
    }
}

```

The *SPCollisionDetection* function takes the result of the *StoSDistance* function to output whether the output of the function means that there is no collision, the point is on the line, or there has been a collision. The expected output of this function should be "No collision", which when ran gave the expected results. Therefore, the function is a success.

```

void Sphere::StoSDistance(Sphere& s1, int s0r, int s1r) { // This function calculates the distance of two spheres

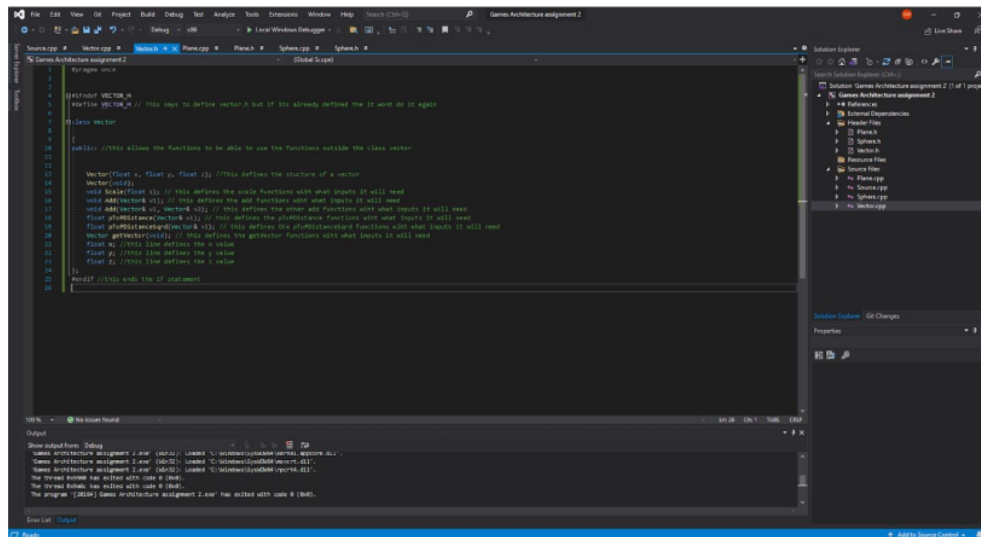
    float Dist = sqrt(pow(x - s1.x, 2) + pow(y - s1.y, 2) + pow(z - s1.z, 2));
    cout << "The distance between the two spheres is: " << Dist << endl;
    float gap = s0r + s1r;

    if (Dist > gap) {
        cout << "No collision detected" << endl;
    }
    else if (Dist == gap) {
        cout << "Objects are adjoint" << endl;
    }
    else if (Dist < gap) {
        cout << "Collision detected" << endl;
    }
}

```

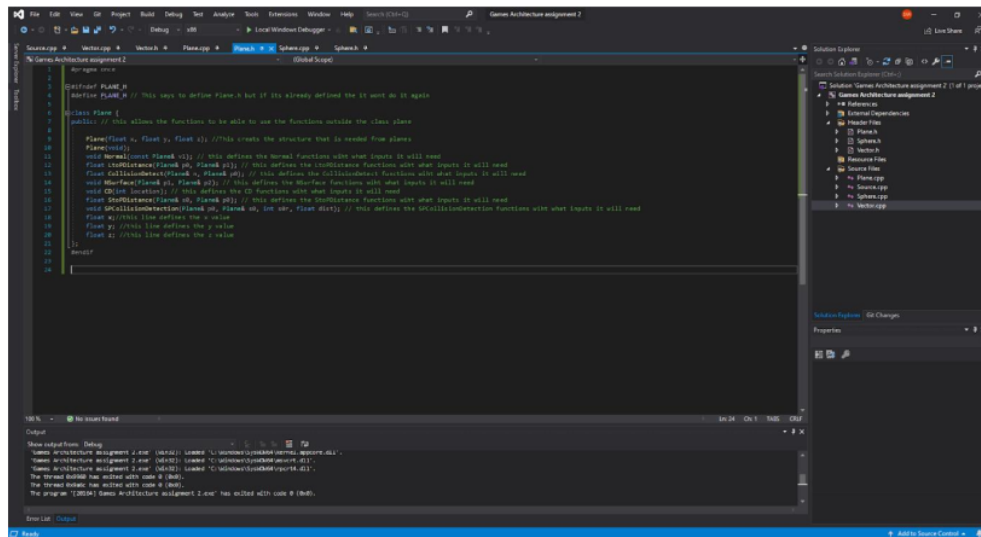
The *StoSDistance* function is made to calculate the distance between two spheres it does this by taking the point in the middle of the spheres and then measuring the distance between both, and finally, it takes the radiuses from both spheres and takes it away from the final distance. This function also declares if there is a collision or not or if the spheres are just touching. The test inputs of this function where one sphere had a point of 1,2, and 3, with a radius of 5 and then the other sphere had the point of 6, 7, and 8 and a radius of 4, the expected result of the distance is a small positive number and the expected result of the collision test is that if the number is below 9 it's a collision if it's 9 then there just touching and if it's higher then there is no collision. The results of this function, are a distance of 8.88 and that "Collision detected"

Vector Class



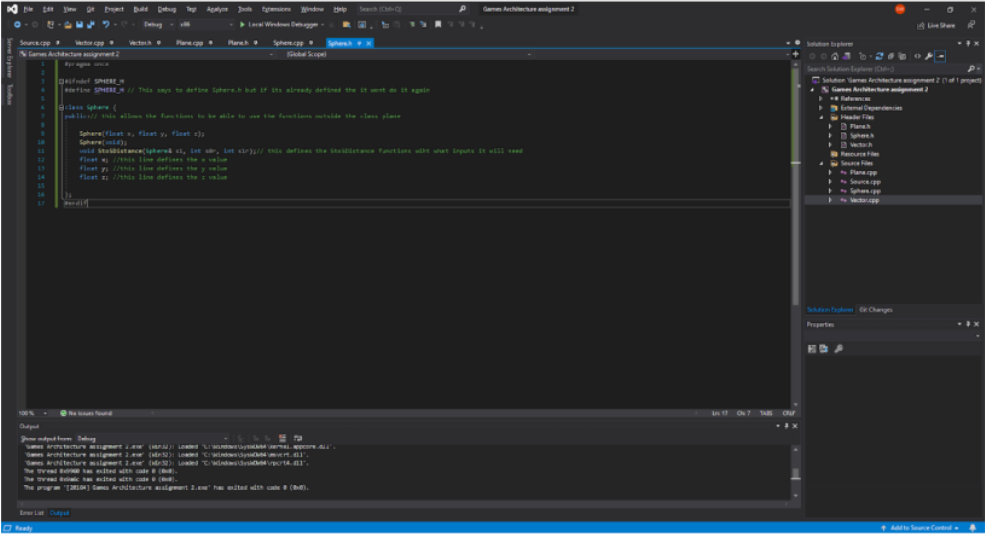
```
1 // Vector.h
2 // This file defines the Vector class.
3
4 #ifndef VECTOR_H
5 #define VECTOR_H
6
7 // This allows the functions to be able to use the functions outside the class vector
8 #include <iostream>
9
10 // This defines the structure of a vector
11 struct Vector {
12     float x; // This defines the x value
13     float y; // This defines the y value
14     float z; // This defines the z value
15 };
16
17 // This defines the constructor function
18 Vector() {}
19
20 // This defines the destructor function
21 ~Vector() {}
22
23 // This defines the Scale function
24 void Scale(Vector v, float s) {}
25
26 // This defines the Add function
27 void Add(Vector v1, Vector v2) {}
28
29 // This defines the Subtract function
30 void Subtract(Vector v1, Vector v2) {}
31
32 // This defines the Multiply function
33 void Multiply(Vector v1, Vector v2) {}
34
35 // This defines the Divide function
36 void Divide(Vector v1, Vector v2) {}
37
38 // This defines the GetX function
39 float GetX(Vector v) {}
40
41 // This defines the GetY function
42 float GetY(Vector v) {}
43
44 // This defines the GetZ function
45 float GetZ(Vector v) {}
46
47 // This defines the Set function
48 void Set(Vector v, float x, float y, float z) {}
49
50 #endif
```

Plane Class



```
1 // Plane.h
2 // This file defines the Plane class.
3
4 #ifndef PLANE_H
5 #define PLANE_H
6
7 // This allows the functions to be able to use the functions outside the class plane
8 #include <iostream>
9
10 // This defines the structure of a plane
11 struct Plane {
12     float x; // This defines the x value
13     float y; // This defines the y value
14     float z; // This defines the z value
15 };
16
17 // This defines the constructor function
18 Plane() {}
19
20 // This defines the destructor function
21 ~Plane() {}
22
23 // This defines the Scale function
24 void Scale(Plane p, float s) {}
25
26 // This defines the Add function
27 void Add(Plane p1, Plane p2) {}
28
29 // This defines the Subtract function
30 void Subtract(Plane p1, Plane p2) {}
31
32 // This defines the Multiply function
33 void Multiply(Plane p1, Plane p2) {}
34
35 // This defines the Divide function
36 void Divide(Plane p1, Plane p2) {}
37
38 // This defines the GetX function
39 float GetX(Plane p) {}
40
41 // This defines the GetY function
42 float GetY(Plane p) {}
43
44 // This defines the GetZ function
45 float GetZ(Plane p) {}
46
47 // This defines the Set function
48 void Set(Plane p, float x, float y, float z) {}
49
50 #endif
```

Sphere Class



Test Program

```
int main(void) {  
  
    //Task 1  
  
    Vector v1(1, 2, 3); //This creates a new Vector  
  
    float dist;  
    Vector v2(2, 3, 4);  
  
    dist = v1.pToPDistance(v2); //this line of code uses the first vector and the second vector and runs it through the  
    ; //pToDistance function to give a distance between two points  
    cout << "The distance between two points is: " << dist << endl;  
  
    float distSqrd;  
    distSqrd = v1.pToPDistanceSqrd(v2);  
    cout << "The distance between two points in square space is: " << distSqrd << endl;  
  
    cout << endl; //Task 2  
  
    Plane P0(3, 4, 5); // Point on the plane  
    Plane Pv(4, 5, 6); // vector of the plane  
  
    Plane n(0, 0, 0); // the normal of the plane  
  
    n.Normal(Pv);  
  
    //n.Normal(Pv);  
    cout << "The normal of the line is: x=" << n.x << ", y=" << n.y << ", z=" << n.z << endl;  
  
    Plane P1(5, 6, 7); // Random point off the plain  
    float ltoPDist;  
    ltoPDist = n.ltoPDistance(P0, P1); //this line of code uses the point on the line, the normal and the point off the line and then runs it through the  
    ; //ltoDistance function to give a distance between a plane and a point  
    cout << "The distance between the point and the plane is: " << ltoPDist << endl;  
  
    float location;  
  
    location = n.CollisionDetect(P0, P1); // This line of code outputs the position of the point from the code  
  
    cout << "The location of the point from the line is: " << location << endl;  
  
    float location;  
  
    location = n.CollisionDetect(P0, P1); // This line of code outputs the position of the point from the code  
  
    cout << "The location of the point from the line is: " << location << endl;  
  
    P1.CD(location);  
  
    Plane P2(9, 4, -4); // Problematic part of code. Dont believe it works  
    Plane AccSurface(0, 0, 0);  
    AccSurface.NSurface(P0, P2); // This line of code tries to calculate the normal from two points  
    cout << "The normal of the line is: x=" << AccSurface.x << ", y=" << AccSurface.y << ", z=" << AccSurface.z << endl;  
  
    cout << endl; //Task 3  
  
    Sphere s0(1, 2, 3);  
    Sphere s1(6, 7, 8);  
  
    int s0r = 5;  
    int s1r = 4;  
  
    float sdist;  
    float Gap = s0r + s1r; //This adds the two radiuses  
  
    s0.StoSDistance(s1, s0r, s1r); //This line uses the StoSDistance function which calculates the distance between the center of two spheres then checks the  
    ; //radius of both to see if there is a collision and then outputs the result
```

```
cout << endl; //Task 4

Plane s3(7, 8, 9);
Plane p3(2, 3, 4);
Plane pv2(3, 4, 5);
Plane spn(0, 0, 0);
spn.Normal(pv2); // this calculate the normal of the new line
float spDist;

float s3r = 5;

spDist = spn.StoPDistance(s3, p3); //This uses the StoPDistance function to calculate the distance between the sphere and the plane
cout << "The distance between the sphere and the plane is: " << spDist << endl;

s3.SPCollisionDetection(p3, s3, s3r, spDist); // This outputs the the results if there is a collision between the sphere and the plane
return 0;
```

Results



```
Microsoft Visual Studio Debug Console
The distance between two points is: 1.73205
The distance between two points in square space is: 3

The normal of the line is: x=0.25, y=0.2, z=0.166667
The distance between the point and the plane is: 3.28126
The location of the point from the line is: 1.23333
No collision
The normal of the line is: x=-36, y=57, z=-24

The distance between the two spheres is: 8.66025
Collision detected

The distance between the sphere and the plane is: 8.88351
No collision detected

C:\Users\steff\source\repos\Games Architecture assignment 2\Debug\Games Architecture assignment 2.exe (process 27052) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Bibliography

- [1] G.Dickers, "Collision Detection1", UWTSD, Swansea, 2022
- [2] G.Dickers, "Tutorial04", UWTSD, Swansea, 2022