

# Assignment 2

*by* Jacob James-MacHin

---

**Submission date:** 11-Jan-2025 10:43AM (UTC+0000)

**Submission ID:** 248023092

**File name:** 50542\_Jacob\_James-MacHin\_Assignment\_2\_628793\_122386092.docx (497.36K)

**Word count:** 2209

**Character count:** 11691

# GAMES ARCHITECTURE AND MECHANICS A2 COLLISION PHYSICS

This paper will go into detail on C++ code and vector integration when it comes to collision simulation. This will also detail the use of sphere collision also as it is the cornerstone of game engine creation

By Jacob  
James-Machin

## Contents

|                                  |   |
|----------------------------------|---|
| Introduction .....               | 2 |
| Investigation.....               | 2 |
| Vector Classes .....             | 2 |
| Distance between two points..... | 2 |
| Mathematical operations.....     | 3 |
| C++ code .....                   | 3 |
| Plane Class .....                | 4 |
| Mathematical operations.....     | 4 |
| C++ code .....                   | 4 |
| Sphere Class with collision..... | 5 |
| Mathematical operations.....     | 5 |
| C++ code .....                   | 5 |
| Point to Plane .....             | 7 |
| Mathematical operations.....     | 7 |
| C++ code .....                   | 7 |
| Conclusion .....                 | 8 |
| Bibliography .....               | 9 |

## Introduction

The aim for this report is to investigate the various uses of a Vector Class using C++ programming language. We will look at the mathematical operations that are used within C++, then develop code and discuss the underlying theoretical concepts that goes into the implementation of the code. The goal will be to list and discuss various required operations using a Vector Based Class System.

Using Vectors is part of Object-Oriented Programming. They can be used to create objects or classes within C++ to help the programmer identify what to work with or what to fix. A Vector is a type of container which can store elements of a similar type [1]

Furthermore, They, can be extremely helpful to store and manipulate points as demonstrated if you are developing using an already established game engine.

To summarize it makes C++ code more manageable by using objects or classes in a manageable way and it helps developers understand what they are working with regarding programming. [2]

## Investigation

To investigate this, A definition on what a Vector Class is and what else is required is to look at the uses of a Vector Class and how its implemented within C++. Another thing to consider regarding coding like this is that the mathematical equations behind these computational operations that make collisions work in the first place. The most important part to understand the method is to understand how Vectors work within the field of mathematics.

To further this investigation, there is a requirement to understand how these engines work in the first place and how they are developed. When developers go out to create their own engine they first choose which language to program in and this language is seen as C++ as it is the most popular language for many game developers looking to build their games as it provide a robust development environment which allows the developers to understand the demands of the computer hardware and capabilities while also allowing for a robust simulation of physics. As will be demonstrated within this investigation into C++ and collision mechanics. [3]

## Vector Classes

Vector Classes are defined through a class template which can be used for sequence containers as they would store distinct types in an arrangement. Vector Classes can allow for fast random access to any element within the program. You can define a vector class through the function called "std::vector" which unlike arrays can account for the size of its elements that are added or removed. [4] It is an immensely powerful container tool in C++.

### Distance between two points.

To utilize this method meaning working out the distance between two points we would use Pythagoras theorem. Basically, you take a line and then turn it on a 90 degrees angle to figure out the length of the line.

## Mathematical operations

The length squared plus the breadth squared. Once added together you would square root them you will end up with three or a “one two three” point triangle. This would only work with a right-angle triangle, and this will help the calculation regarding the size of the line. The mathematical equations for this are...  $a^2 + b^2 + c^2 = c^2$

## C++ code

Here is the code to help calculate the distance between two points. First thing required is to create a function that allows for the calculation to occur for instance within the code there is an implementation that starts with making the distance between the two vectors into a “Float” which would make the distance between the vectors into a value that would change depending on the output.

```
int main()
{
    Vector v1(01, 02, 03); //This is the Vector's v1 initial position
    float dist; //The distance would be a float
    Vector v2(02, 03, 04); //like v1 this is for initial position for v2
    dist = v1.PtoPDistance(v2); //returns a float that gives the distance between two points
    std::cout << "Where the distance between two points " << dist; //this prints the two distances out
    float distSqrd; //Distance Squared
    distSqrd = v1.PtoPDistanceSqrd(v2); //PtoPDistance essentially means Point to Point distance and using the square root I can calculate the distance effectively

    //The rest of the vector function allows for printing
    std::cout << "Where are the distance between two points squared " << distSqrd;
    std::cout << "Point Distance " << dist << distSqrd;
    v1.print();
    v2.print();
}
```

*Int main code*

Next thing required is to implement the calculation. Using the definition created in the main file it would allow the program to understand where it will take the calculation.

```
19 float Vector::PtoPDistance(Vector v2) //Point to Point distance calculation
20 {
21     return std::sqrt(std::pow(v2.x - x, 1) + std::pow(v2.y - y, 1) + std::pow(v2.z - z, 1));
22 }
23 return 0.0f;
24
25 float Vector::PtoPDistanceSqrd(Vector v2) //Point to point squared. This helps both the vectors and the sphere collisions
26 {
27     return std::sqrt(std::pow(v2.x - x, 2) + std::pow(v2.y - y, 2) + std::pow(v2.z - z, 2));
28     return 0.0f;
29 }
30
31
```

*Floats for the PtoP Distance*

This part of the calculation tells the program to implement the Distance between two points which would detect the collision and the distance between those  points. Using the x, y, z signifiers allow to determine where it would collide with vectors using the square root of the distance.

Also, the next important part is this is the Header File which would allow for the establishment of the various functions within the program for example creating the float functions and which role they would play within the program, or the header file can also help establish the various Classes used within the function. In this case the utilization of the Vector classes where these functions would be utilized.

```

1  #pragma once
2  #include <cmath>
3
4
5
6  class Vector
7  {
8  public:
9
10     //These are constructors
11     Vector(float x, float y, float z);
12     Vector(void);
13
14     //These are operators
15     void Scale(float s);
16     void Add(Vector& v1);
17     void Add(Vector& v1, Vector& v2);
18     Vector getVector(void);
19
20     float PtoPDistance(Vector v2); //point to point distance
21     float PtoPDistanceSqrd(Vector v2);
22
23     //new implementations for the collision detection
24
25     float x;
26     float y;
27     float z;
28
29     void print();
30
31 };

```

As demonstrated, there is a Class Vector where the various functions used to calculate the Point-to-Point calculations and the x, y, z as a float function and finally the print function that can be used to display this function in an output window.

## Plane Class

Within C++ or any other languages, you can determine the physics of various objects using the mentioned Vector Classes. For example, determining the physics of a plane. They can allow you to calculate the distance regarding the front or the back of the plane. Using the established Vector Classes within the code.

## Mathematical operations

To define a plane equation, you would need a point and a vector that helps describe the line. You could start from points b and a and take  $v = a - b$ . Within the code its essentially trying to determine the surface of an area through  $a*x + b*y + c*z + d = 0$

## C++ code

```

1  #include "vector.h"
2  #include <iostream>
3
4  //this is a constructor
5  Vector::Vector(void) : x(0), y(0), z(0) {};
6  Vector::Vector(float myx, float myy, float myz)
7  {
8      x = myx;
9      y = myy;
10     z = myz;
11 }

```

*Constructor for the Vectors*

Part of the Vector class this part of the code you can determine the plane class of the said Vector class. This will help calculate the plane class again using the x, y, z. Doing this would help if you were using a constructor which would help build the plane class for the calculation. This will help calculate the distance between these two points also since it establishes that as the method for the dot product of the two vectors.

## Sphere Class with collision.

The Idea behind Collision is physics based. It is a short duration interaction between physical bodies causing change in said bodies. This process usually involves a change in velocity and momentum and the velocity impact is seen as the "Closing Speed." There are differing types of Collision such as Elastic Collision (which involves kinetic energy meaning Objects would bounce off each other) and Inelastic Collision (Basically if all the kinetic energy is lost the objects will lose momentum and stop dead in its tracks) [5]

## Mathematical operations

The maths behind Collisions involves Momentum (p), Mass (m) and Velocity (v). The idea behind this equation is to detect the intersection of objects in a space. Understanding this concept will help with applications like computer graphics, physical simulation, video games and robotics [5] to calculate the act of collision there needs to be a determination of the law of conservation of momentum. It is linked to calculating the both the objects momentum, mass, and velocity. It is understood through  $p + m + v$  where the speed and momentum of the objects is determined by the mass (which means weight of the object in this context

## C++ code

To simulate this in code it can take some time to understand the intricacies however if the code already a jumping off point with the Vectors has calculation then it would be rather simple to understand how this concept would apply to 3D objects like spheres and how they would interact.

```
27 //Spheres are created
28 SphereCollision sphere1(Vector(0, 0, 0), 5.0f);
29 SphereCollision sphere2(Vector(8, 0, 0), 5.0f);
30 SphereCollision sphere3(Vector(12, 0, 0), 5.0f);
31
32 //this prints the message determining the spheres colliding
33 std::cout << "Sphere 1 and Sphere 2: "
34 << (sphere1.isColliding(sphere2) ? "Colliding" : "Not Colliding") << std::endl;
35
36 std::cout << "Sphere 1 and Sphere 3: "
37 << (sphere1.isColliding(sphere3) ? "Colliding" : "Not Colliding") << std::endl;
38
39
40 sphere1.print();
41 sphere2.print();
42 sphere3.print();
43
44 return 0;
45
```

### Sphere Collisions

To start this is to establish the spheres that the program would use. Then use a value to determine the size and the collision. Which would allow for the program to calculate the spheres. Theres also the print function which would print the output into the terminal that would be displayed through the Operating System's command prompt.

The next part of the code involves.

```
19  float Vector::PtoPDistance(Vector v2) //Point to Point distance caculation
20  {
21      return std::sqrt(std::pow(v2.x - x, 1) + std::pow(v2.y - y, 1) + std::pow(v2.z - z, 1));
22  }
23  return 0.0f;
24  }
25  float Vector::PtoPDistanceSqr(Vector v2) //Point to point squared. This helps both the vectors and the sphere collisions
26  {
27      return std::sqrt(std::pow(v2.x - x, 2) + std::pow(v2.y - y, 2) + std::pow(v2.z - z, 2));
28  }
29  return 0.0f;
30  }
31  }
```

*Floats for the PtoP distance 2*

Is adding the Sphere Collision equation to the Point to Plane distance equations which is important to determine how far the Spheres are moving away from each other which would help with the next step in the equation.

```
37
38  //This is the SphereCollision constructor
39  SphereCollision::SphereCollision(Vector center, float radius)
40  { : center(center), radius(radius) {}
41  }
42  //To detect this new collision method
43
44  bool SphereCollision::isColliding(SphereCollision& other) //For the Sphere Collision I've used a bool function
45  {
46      float distSqr = center.PtoPDistanceSqr(other.center);
47      float radiusSum = radius + other.radius;
48      return distSqr <= (radiusSum * radiusSum);
49  }
50  }
51
52  //To print the final results
53
54  void SphereCollision::print() //this prints the collision function out along with the vector collision
55  {
56      std::cout << "\nSphere Center: (" << center.x << ", " << center.y << ", " << center.z << ")";
57      std::cout << ", Radius: " << radius << std::endl;
58  }
```

*Sphere Collision bool*

The creation of the constructor would help build the equations which would determine the center of the sphere and the radius. After which there is also making the collision into a bool or Boolean which asks the program whatever if the Spheres are colliding or nor. After which there is an effective method which involves in creating a float function for the distance that is related to the center and the radius of the sphere.

Finally, for both the Vectors and Spheres the header file is an important part of the creation of this program.

For the Spheres collision header file.

```

33 class SphereCollision
34 {
35 public:
36     Vector center;
37     float radius;
38     SphereCollision(Vector center, float radius);
39     bool isColliding(SphereCollision& other);
40     //bool isColliding(const SphereCollision& other) const;
41     //bool isColliding(const SphereCollision& other) const;
42
43     void print();
44
45 };
46

```

*Classes for the Sphere Collision*

This involves establishing the center as the vector which would help develop the equations involved in this collision detection system. There's also the radius of the Spheres and how they would collide and how it would relate to the center as a vector. Along with making the collision as a bool which would ask whatever if the spheres were colliding or not.

## Point to Plane

Point to Plane involves the distance between the point and the perpendicular distance to the plane. This involves taking the sphere created for collision detection and using physics simulation to determine where the sphere will collide with. This method is used a lot within 3D game engines as a basis for simulating physics. For instance, when playing a game with a physics engine it's where an object is represented by single point (Like a projectile) and where it collides with on the plane. This is also used in a lot of game engines or even ray tracing. (How light would reflect or interact with the plane) [6] [7]

## Mathematical operations

The maths behind this is typically defined by the equation  $Ax + By + Cz + D = 0$

Where (A, B, C) is the vector of the plane (related to the surface) and D helps the position of the plane on the surface.

A point is represented as P (Xp, Yp, Zp) where XYZ is part of the 3D space (where you calculate the position of where the point would be) within game engines it would be used to determine how the physics work and how would objects would relate to each other.

Another way to simulate this is for example if a Point is above or below the plane the mathematical operation would detect if there would be a collision, and the position of the object would be adjusted accordingly or is the point moves along the plane the physics engine would calculate the reaction (like if you are bouncing) you can calculate this using friction or kinetic energy. [8]

For example, the Source 2 engine uses an advanced form of physics interaction when it comes to collision. Where you can place objects within an object without the program glitching in unexpected ways (like placing a virtual object inside of a virtual box) [9]

## C++ code

To represent this in C++ we use

```

19  float Vector::PtoPDistance(Vector v2) //Point to Point distance caculation
20  {
21      return std::sqrt(std::pow(v2.x - x, 1) + std::pow(v2.y - y, 1) + std::pow(v2.z - z, 1));
22  }
23  return 0.0f;
24  }
25  float Vector::PtoPDistanceSqr(Vector v2) //Point to point squared. This helps both the vectors and the sphere collisions
26  {
27      return std::sqrt(std::pow(v2.x - x, 2) + std::pow(v2.y - y, 2) + std::pow(v2.z - z, 2));
28      return 0.0f;
29  }
30  }
31

```

*Floats for the PtoP distance 3*

As mentioned previously there needs to be an element of calculation within the code with how these objects would interact with each other. As demonstrated within this screenshot there is a use of utilizing the Point-to-Point distance function to calculate the both the vectors colliding with each other and the sphere colliding with each other.

## Conclusion

To conclude this document the idea behind collision is key for games development especially in designing a games engine. As the vast majority of game engines on the market especially modern game engines uses an element of physics to it [10] And its important to understand the very core basics of collision mathematics in order to implement these complex mathematical equations into a more understandable coding format. Such as c++.

## Bibliography

- [1] ""Creating a Vector of Class Objects in C++", " GeeksforGamers, 08 November 2022.  
] [Online]. Available: <https://www.geeksforgeeks.org/creating-a-vector-of-class-objects-in-cpp/>. [Accessed 09 01 2025].
- [2] J. Horton, ""Using C++ Vectors", " Game Code School, 2023. [Online]. Available:  
] <https://gamecodeschool.com/c-plus-plus/using-c-vectors/>. [Accessed 09 12 2024].
- [3] Stefan, "Graphics and Simulation," teamcubate, 10 11 2023. [Online]. Available:  
] <https://teamcubate.com/blogs/how-is-c-plus-plus-used-in-game-development#:~:text=C%2B%2B%20provides%20advanced%20networking%20capabilities,unparalleled%20in%20terms%20of%20performance..> [Accessed 11 01 2025].
- [4] ""vector - C++ Reference," cplusplus.com, [Online]. Available:  
] <https://cplusplus.com/reference/vector/vector/>. [Accessed 01 10 2025].
- [5] Various, "Collision," Wikipedia, 07 17 2020. [Online]. Available:  
] <https://en.wikipedia.org/wiki/Collision>. [Accessed 10 01 2025].
- [6] ""Distance Between Point and Plane - Formula, Proof, Examples," " Cuemath, [Online].  
] Available: <https://www.cuemath.com/geometry/distance-between-point-and-plane/>.  
[Accessed 10 01 2025].
- [7] W. Contributors, ""Ray Tracing Graphics", " Wikipedia, 27 01 2020. [Online]. Available:  
] [https://en.wikipedia.org/wiki/Ray\\_tracing\\_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics)). [Accessed 10 01 2025].
- [8] ""Distance Between Point and Plane- Formula, Proof, Examples," Cuemath, [Online].  
] Available: <https://www.cuemath.com/geometry/distance-between-point-and-plane/>.  
[Accessed 10 01 2025].
- [9] S. Migdalskiy, ""Physics for Game Programmers: Debugging Physics, ", "  
] [www.gdcvault.com](http://www.gdcvault.com), 2015. [Online]. Available:  
<https://www.gdcvault.com/play/1020065/Physics-for-Game-Programmers-Debugging>.  
[Accessed 10 01 2025].
- [1] D. M. Bourg, Physics for Game Developers, Beijing ; Cambridge : O'Reilly & Associates,  
0] 2002.
- [1] [Online]. Available: <https://www.geeksforgeeks.org/creating-a-vector-of-class-objects-in-cpp/>.

[1 [Online]. Available: . <https://gamecodeschool.com/c-plus-plus/using-c-vectors/>.  
2]

# Assignment 2

---

## GRADEMARK REPORT

---

### FINAL GRADE

60/100

### GENERAL COMMENTS

A really great attempt, you were missing a handful of things that would have really pushed this above and beyond!

In the future, when demonstrating your code also show the input and output of your testing, to demonstrate working functionality.

But you smashed it. Great work Jacob.

---

PAGE 1

---

PAGE 2

---

PAGE 3

---



#### Comment 1

In this context we are discussing the vectors in 3d space, i.e. vertices

---

PAGE 4

---



#### Comment 2

why is the D capital

---

PAGE 5

---

PAGE 6

---

PAGE 7

---



#### Comment 3

why did you square radiussum?

---

PAGE 8

---



#### Comment 4

Just for feedback, it is preferred to delete unused or commented-out code, especially when demonstrating them if they do not serve a purpose.

It is what we would call housekeeping

PAGE 9

---



### Comment 5

the first implementation of the ptopdistance here is raising to the power of 1?



### Comment 6

the second return that returns 0.0f is completely unnecessary

PAGE 10

---

PAGE 11

---