

# Assignment 1 Jordan James

## 1803023

*by* Jordan James

---

|                |                                                                                   |                 |      |
|----------------|-----------------------------------------------------------------------------------|-----------------|------|
| FILE           | 11909_JORDAN_JAMES_ASSIGNMENT_1_JORDAN_JAMES_1803023_234537_2658938.DOCX (28.11K) |                 |      |
| TIME SUBMITTED | 07-FEB-2020 10:53PM (UTC+0000)                                                    | WORD COUNT      | 964  |
| SUBMISSION ID  | 119779839                                                                         | CHARACTER COUNT | 4928 |

2020

# Assignment 1 Game Physics Report

BY JORDAN JAMES

## Contents

|                                                           |    |
|-----------------------------------------------------------|----|
| Introduction .....                                        | 2  |
| Common numerical integration methods used in gaming ..... | 3  |
| Euler's Method.....                                       | 3  |
| Verlet Method.....                                        | 3  |
| Vector Class.....                                         | 4  |
| Old Vector File cpp.....                                  | 4  |
| Old Vector file h .....                                   | 5  |
| New Vector File cpp .....                                 | 6  |
| New Vector File h.....                                    | 9  |
| What was modified.....                                    | 10 |
| Testing Table.....                                        | 11 |

## Introduction

In this report I'm going to cover common numerical integration methods used in gaming and their advantages and disadvantages. I will then investigate underlying mathematical theories to identify the appropriate methods to use for a vector class used in a game engine. I will talk about the functions I added and functions I removed in detail. I will then provide a test table containing my tests on the class file.

## Common numerical integration methods used in gaming

### Euler's Method

Euler's Method allows you to calculate the velocity at a specific time point and estimate what the next velocity will be. Though this is the simplest method to add, it's not as accurate as other methods. Euler's Method has a higher chance of getting the estimation wrong the larger the time spot difference is. For example, if you were to go from 1 second to 1.5 seconds, it would probably be ok. But 1 second to 100 seconds is a large difference. This makes Euler's method 1<sup>st</sup> order accurate. It would be considered an Advantage to be easily implemented, but a disadvantage to have less accuracy.

### Verlet Method

There are different types of the Verlet method. Position, Leapfrog and Velocity. For this, when I refer to Verlet Method, I mean the Verlet Position Method. The Verlet Method is almost as fast as Euler's Method, which is the Verlet's downfall, it's slower than other methods. But it makes up for it with a very accurate 3<sup>rd</sup> - 4<sup>th</sup> order accurate. The Verlet Method also requires 2 initial time points before it starts. It also handles change in time points badly.

## Vector Class

Old Vector File.cpp

```
#include "vector.h"
//construct
Vector::Vector(void){};
Vector::Vector(float myx, float myy, float myz)
{
    x= myx;
    y= myy;
    z= myz;
}
void Vector::Scale(float s)
{
    x= s*x;
    y= s*y;
    z= s*z;
}
void Vector::Add(Vector & v1, Vector & v2)
{
    v1.x= v2.x + v1.x;
    v1.y= v2.y + v1.y;
    v1.z= v2.z + v1.z;
}
Vector Vector::getVector(void)
{
    return Vector(x,y,z);
}
void Vector::Add(Vector & v1)
{
    x= x + v1.x;
    y= y + v1.y;
    z = z + v1.z;
}
```

Old Vector file h

**/\*\*\*\*\*(note x,y,z are not private--should they be?)**

**#ifndef VECTOR\_H**

**#define VECTOR\_H**

class Vector

{

public:

    //construct

    Vector(float x, float y, float z);

    Vector(void);

    void Scale(float s);

    void Add(Vector & v1);

    void Add(Vector & v1, Vector & v2);

    Vector getVector(void);

**//private:   //these are currently public – should they be private?**

    float x;

    float y;

    float z;

};

**#endif**

New Vector File cpp

```
#include "vector.h"
```

```
/* Constructor
```

```
    @Params: void
```

```
    @Description: Allow class to be referenced
```

```
*/
```

```
Vector::Vector(void) {};
```

```
/* Constructor
```

```
    @Params: myx: float, myy: float, myz: float
```

```
    @Description: Takes in 3 parameters and sets the objects respected  
variables to each float.
```

```
*/
```

```
Vector::Vector(float myx, float myy, float myz)
```

```
{
```

```
    this->x = myx;
```

```
    this->y = myy;
```

```
    this->z = myz;
```

```
}
```

```
/* Scale
```

```
    @Params: s: float
```

```
    @Description: Scales up a vector by a specific increment.
```

```
    @returns: void
```

```
*/
```

```
void Vector::Scale(float s)
```

```
{
```

```
    this->x = s * this->x;
```

```
    this->y = s * this->y;
```

```
    this->z = s * this->z;
```

```
}
```

```
/* getMagnitude
```

```
    @Params: N/A
```

```
    @Description: Calculates the length of a vector using a formula.
```

```
    @returns: float
```

```
*/
```

```
float Vector::getMagnitude() {
```

```
    return std::sqrtf((this->x * this->x) + (this->y * this->y) + (this->z * this->z));
```

```
}
```

```
/* pToPDistance
```

```
    @Params: * v1: Vector
```

```
    @Description: Calculates the distance between this object and the vector  
parsed into the parameters.
```

```
    @returns: float
```

```
*/
```

```
float Vector::pToPDistance(Vector * v1) {
```

```
    return std::sqrtf((this->x - v1->getX()) * (this->x - v1->getX()) +
```

```

        (this->y - v1->getY()) * (this->y - v1->getY()) +
        (this->z - v1->getZ()) * (this->z - v1->getZ()));
    }

    /*      Add
       @Params: & v1: Vector
       @Description: Add current object variables with the vector parsed into the
       parameters' variables.
       @returns: void
    */
    void Vector::Add(Vector & v1)
    {
        this->x = this->x + v1.x;
        this->y = this->y + v1.y;
        this->z = this->z + v1.z;
    }


    /*      GETTERS & SETTERS
       @Params: x: float, y: float, z: float
       @Description: functions to allow read and write on Vector classes private
       variables
       @returns: float, float, float, void, void, void
    */

    Vector Vector::getVector(void)
    {
        return Vector(this->x, this->y, this->z);
    }
    float Vector::getX()
    {
        return this->x;
    }
    float Vector::getY()
    {
        return this->y;
    }
    float Vector::getZ()
    {
        return this->z;
    }
    void Vector::setX(float x)
    {
        this->x = x;
    }
    void Vector::setY(float y)
    {
        this->y = y;
    }

```

```
}  
void Vector::setZ(float z)  
{  
    this->z = z;  
}
```

New Vector File h

#pragma once

#include <iostream>

#include <string>

#include <sstream>

class Vector {

public:

Vector(void);

Vector(float x, float y, float z);

void Scale(float s);

float getMagnitude();

float pToPDistance(Vector \* v1);

void Add(Vector & v1);

Vector getVector(void);

float getX();

float getY();

float getZ();

void setX(float x);

void setY(float y);

void setZ(float z);

private:

float x;

float y;

float z;

};

### What was modified

I added one function to the vector class, called `ptoPDistance`. This function takes in a single vector parameter. It then returns the square root of  $\text{vector1.x} - \text{vector2.x}$  squared. Plus  $\text{vector1.y} - \text{vector2.y}$  squared. Plus  $\text{vector1.z} - \text{vector2.z}$  squared. I also removed a function as I didn't feel like it would be of any use to the class. The function was one of the Add functions. There already exists a function that takes in one vector parameters and adds that against the selected object. Having another function that takes in 2 vector parameters and adding them together would be unneeded.

### Testing Table

| Test Number | Function     | Input    | Expected Result | Actual Result | Pass / Fail |
|-------------|--------------|----------|-----------------|---------------|-------------|
| 1           | ptoPDistance | 1, 2, 3  | 0               | 0             | Pass        |
| 2           | ptoPDistance | 5, 6, 7  | 6.928203        | 6.928203      | Pass        |
| 3           | ptoPDistance | 7, 3, -6 | 10.86278        | 10.86278      | Pass        |

# Assignment 1 Jordan James 1803023

## GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

**Instructor**

33/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4



### Comment 1

1sec is a very large difference - even millisecond differences would show errors

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

GRADING FORM: SMU GENERIC\_GJ

JORDAN  
JAMES

32.50

#### RESEARCH-INFORMED LITERATURE



Evidence of investigation and research but use of references at all. You must include refs/bibliography in your report.

3.5

#### KNOWLEDGE AND UNDERSTANDING



Report shows some understanding of Integration methods but lacks detail. Code covers only point to point dist calc.

7

#### ANALYSIS



A basic discussion of Integration methods - could be clearer. No results to analyse for task one. Task 2 point to point method implemented and tested.

7

#### PRACTICAL APPLICATION AND DEP.



Point to point distance implemented - ok. No evidence of coding Euler or other tasks.

11

#### SKILLS FOR PROFESSIONAL PRAC.



No refs. Report structured.

**4**