

GAM Assignment Daniel O'Dwyer 1801164

by Daniel O'dwyer

FILE	12011_DANIEL_ODWYER_GAM_ASSIGNMENT_DANIEL_ODWYER_1801164_234537_2063849709.DOCX (2.37M)		
TIME SUBMITTED	07-FEB-2020 11:52PM (UTC+0000)	WORD COUNT	2048
SUBMISSION ID	119782167	CHARACTER COUNT	10093

Games Architecture and Mechanics

[ACCA5027]

By

Daniel O'Dwyer 1801164

Table of Contents

- ❖ Euler's Method [page 3]
- ❖ Numerical Integration in Games [pages 3 - 4]
- ❖ Verlet Method [pages 4 - 5]
- ❖ Developed Algorithm [pages 5 - 8]
- ❖ Sphere Collision [pages 8 - 11]
- ❖ Distance Between Points [pages 11 - 14]
- ❖ Planes [pages 15 - 19]
- ❖ Bibliography [page 20]

Euler's Method

Euler's method is a numerical integrator, it is one of the more widely known ones and offers a straightforward approach. Its main purpose is to find solutions to differential equations by showing them as a curve with line segment points. Euler's method is most useful in physics as often the exact answer to a differential equation cannot be obtained, this is where Euler's method shines. Instead of a pinpoint result this method gives us an approximation. While the result is still not a definite answer, it's very close. Euler's method uses a starting point and attempts to find another value close to it. As the x value increases the y also does but by a certain amount depending on the change in x .

As explained by Brilliant, "Suppose we have a function $f(x)$, and we know the value of f at a point x_0 , and $f'(x_0)$. If we want to find the value of f at a point close to x , say $x+h$, then we can approximate $f(x+h)$ by using the tangent line to the graph at the point $(x_0, f(x_0))$, as shown in the image to the right. The idea is that $f'(x_0)$ estimates how much change there is in y for each change in x ; we know that x increased by h , so y should have increased by approximately $hf'(x_0)$, so the new value should be $f(x_0) + hf'(x_0)$." [1]. As the h value gets larger the result will become less accurate, this method uses tangent line approximations constantly to fix this. Euler's Method can also be used to solve first order differential equations. Euler's method has a computing time step of $O(h)$. This doesn't sound bad, but when compared with other integrators its quite inefficient. As a basic integration method, it's easy to understand making it an excellent starting point for anyone new to integrators.

Advantages

- Basic and simple integration method which makes it easy to understand

Disadvantages

- Results can get extremely inaccurate unless small steps in each calculation are taken
- Small step size can lead make it extremely inefficient
- Accuracy is poor compared to other integration methods

Numerical Integration in Games

Semi-Implicit Euler: This method is somewhere in between standard Euler's method and back Euler. It takes the best of both methods and uses them to calculate velocity, position and acceleration. It uses standard Euler's method to calculate the velocity and acceleration while using back Euler's method to calculate position. Its use in gaming would be primarily for physics-based movement calculations. In particular it would be used for rigid body movement.

Advantages

- Simple integration method which makes it easy to understand
- More stable than standard Euler's method
- Very popular in games development

Disadvantages

- Accuracy is still poor compared to other integration methods, however it is slightly improved

Velocity Verlet: Velocity Verlet seeks to improve on the standard Verlet method by adding more steps to offer a more accurate calculation of velocity. This method would be most commonly used in racing games where drastic changes in speed are common place, thus accurate calculations are needed. While its efficiency is lacklustre, the accuracy more than makes up for it.

Advantages

- Accurate calculations, especially in velocity
- Commonly used in games
- Good balance between efficiency and simplicity

Disadvantages

- Less efficient than most integrators
- More steps mean more load on CPU processing

Verlet Method

Verlet method is a second order integrator, it's a popular integrator in the gaming industry. Being a second order integrator means that it possesses greater accuracy than its first order counterparts. According to James M. Van Verth and Lars M. Bishop, Verlet method, "is commonly used in molecular dynamics" [2] and "can simulate soft objects such as cloth, rope and dead bodies." [2]. When compared with Euler's method Verlet offers more accuracy while retaining its simplicity. These merits make it a popular choice in the gaming industry as mentioned above. It also retains an efficiency that most second order integrators do not. Euler's method can become easily overloaded, Verlet method keeps its cool and offers accurate approximations without the risk of overloading. However, while Euler's method can approximate position, acceleration and velocity equally well, Verlet favours position and acceleration while velocity gets left in the dust. Luckily it can amend this drawback by using its related method Velocity Verlet, which improves its velocity calculation to an acceptable standard.

Advantages

- Accurate calculations
- Very efficient
- Fast calculation times

Disadvantages

- Velocity approximation is sorely lacking
- Cannot work well with varied time steps

Developed Algorithm

Euler's method was used for the first iteration of the developed algorithm. The algorithm contains some constant values that are used in calculations such as mass, rolling resistance and gravity. These constants are defined in the class that the algorithm uses, there are other values like velocity, time and acceleration which must be calculated in the algorithm. They will change so they are not constants, thus they are not defined with values in the class file. The time increases by 0.06 seconds each step, the program runs until the velocity ends with a value of 0. Verlet method follows the same structure with minor alterations to the code. The code for Euler's method looks like this,

```
void Car::Euler()
{
    /* Fd = drag force */
    /* Ft = tractive force */

    velocity = 0.0;
    dragF = 0.0;
    tracF = 0.0;
    time = 0.0;
    accel = 0.0;
    distance = 0.0;

    while (velocity > 0 || time <= 0.06)
    {
        if (velocity < 0)
        {
            dragF = 0;
        }
        else
        {
            dragF = (RollingRes * mass * gravity) + (0.5 * dragCo * airdensity * frontArea * pow(velocity, 2));
        }

        distance = distance + (velocity * time);

        if (distance > 0)
        {
            tracF = 0;
        }
        else
        {
            tracF = 1.5 * (16 - distance);
        }

        accel = (tracF - dragF) / mass;
        time += 0.06;

        cout << "Time: " << time << '\t' << "Distance: " << distance << '\t' << " Velocity: " << velocity << endl;
        velocity = velocity + (accel * time);
        dragF = (RollingRes * mass * gravity) + (0.5 * dragCo * airdensity * frontArea * pow(velocity, 2));
        tracF = 1.5 * (16 - distance);
        accel = (tracF - dragF) / mass;
    }
}
```

The class looks like this,

```
class Car
{
public:
    const float RollingRes = 0.21;
    const float mass = 1;
    const float gravity = 9.81;
    const float dragCo = 0.34;
    const float airdensity = 1.2;
    const float frontArea = 0.01;
    float velocity;
    float dragF;
    float tracF;
    float time;
    float accel;
    float distance;
    void Euler();
    void Verlet();
};
```

The code for Verlet method looks like this,

```
void Car::Verlet()
{
    /* Fd = drag force */
    /* Ft = tractive force */

    velocity = 0.0;
    dragF = 0.0;
    tracF = 0.0;
    time = 0.0;
    accel = 0.0;
    distance = 0.0;

    while (velocity > 0 || time <= 0.06)
    {
        if (velocity < 0)
        {
            dragF = 0;
        }
        else
        {
            dragF = (RollingRes * mass * gravity) + (0.5 * dragCo * airdensity * frontArea * pow(velocity, 2));
        }

        distance = distance + (distance - distance) + (accel * (pow(time, 2)));

        if (distance > 8)
        {
            tracF = 0;
        }
        else
        {
            tracF = 1.5 * (16 - distance);
        }

        accel = (tracF - dragF) / mass;

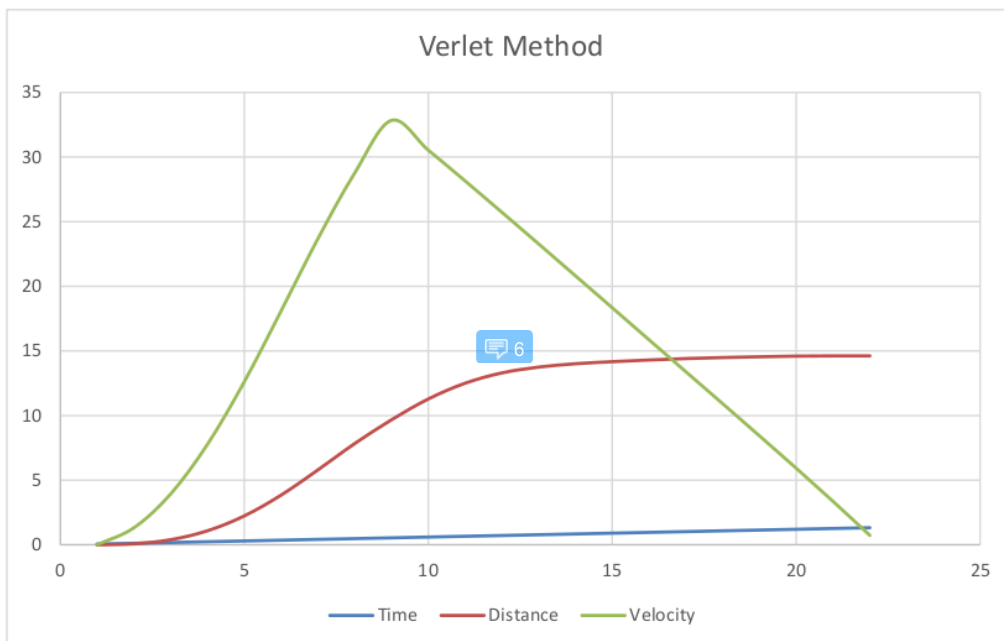
        time += 0.06;

        cout << "Time: " << time << '\t' << "Distance: " << distance << '\t' << "Velocity: " << velocity << endl << endl;

        velocity = velocity + (accel * time);

        dragF = (RollingRes * mass * gravity) + (0.5 * dragCo * airdensity * frontArea * pow(velocity, 2));
        tracF = 1.5 * (16 - distance);
        accel = (tracF - dragF) / mass;
    }
}
```

Graphs



When we compare the two graphs, there is a noticeable difference. Euler's graph has a significantly bigger increase in distance than Verlet. However, the velocity is considerably lower. The time is also shorter which means Verlet has more values on its graph. Verlet has a staggering increase in velocity when compared to Euler's, it also has a steeper incline and decline representing a quick spike rather than a gradual increase. The car in Verlet didn't travel as far as Euler's and had a much steadier distance compared to Euler's constant increase. These results are interesting, Euler's seems to offer a shorter running time with a longer distance. Verlet offers a more thorough result with a steady, constant distance rather than the overall increase we see in Euler's.

Sphere Collision

The program features a simple collision detection method between two spheres. It uses a basic calculation to check that the combined radius isn't bigger than the distance between the two spheres. It's a simple, almost brute force, method but a more complicated method may have caused more issues. As said by Christer Ericson, "Implement a simple reference algorithm (such as a brute-force algorithm that tests all objects or all polygons against each other) and run the reference algorithm in parallel with the more sophisticated algorithm. If the results differ, there is a problem (in all likelihood with the more advanced algorithm)." [3]. The collision detection uses the distance between two points formula which is, $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$. And an if statement to check if the calculated distance is less or equal to the combined radius. In Code form it looks like this,

```
void Sphere::Collision(Sphere &Sphere2)
{
    /* Distance between two points calculation */
    float distance = (pow(pow((x - Sphere2.x), 2) + pow((y - Sphere2.y), 2) + pow((z - Sphere2.z), 2), 0.5));

    /* If statement checks if the combined radius is bigger than the distance between the two spheres */
    if (r + Sphere2.r >= distance)
    {
        /* If the combined radius is bigger then a collision has happened */
        cout << "Collision detected!" << endl << endl;
    }
    else
    {
        /* If the combined radius is smaller then a collision has not happened */
        cout << "No collision detected, we're safe!" << endl;
    }
}
```

The class that it uses looks like this,

```

class Sphere: Vector
{
public:
    Sphere(float myx, float myy, float myz, float myr);
    void Collision(Sphere &sphere2);
private:
    float r;
};

```

Test 1

Description: In this test the values for sphere one are x=4, y=10, z=1 and the radius is 7.5. The values for sphere two are x=5, y=6, z=2 and the radius is 9.

Expected Result: Collision detected.

Actual Result: Collision detected.

Pass/Fail: Pass

```

C:\Users\Dan\Documents\Uni Year2\Assignments>
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 2

Collision detected!

1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: _

```

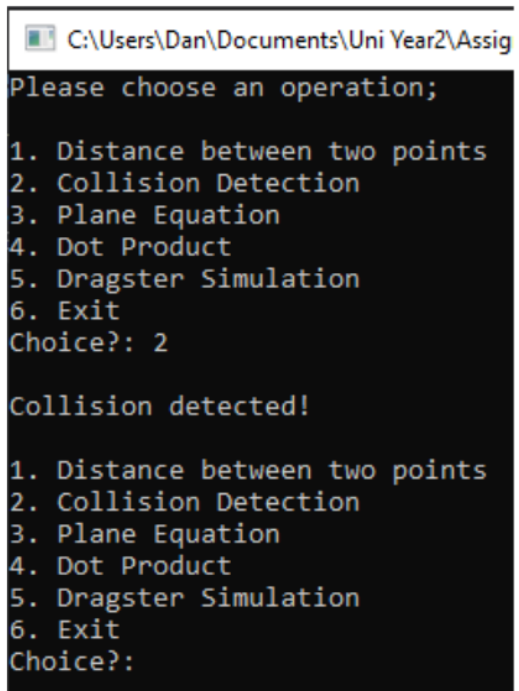
Test 2

Description: In this test the values for sphere one are $x=5$, $y=8$, $z=7$ and the radius is 3.2. The values for sphere two are $x=11$, $y=3$, $z=15$ and the radius is 12.4.

Expected Result: Collision detected.

Actual Result: Collision detected.

Pass/Fail: Pass



```
C:\Users\Dan\Documents\Uni Year2\Assig
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 2

Collision detected!

1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?:
```

Test 3

Description: In this test the values for sphere one are $x=5$, $y=14$, $z=2$ and the radius is 10.8. The values for sphere two are $x=4$, $y=2$, $z=20$ and the radius is 7.6.

Expected Result: No collision detected.

Actual Result: No collision detected.

Pass/Fail: Pass

```
C:\Users\Dan\Documents\Uni Year2\Assignmei
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 2

No collision detected, we're safe!

1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?:
```

Distance Between Points

This method allows us to calculate the exact distance between two points in 3D space. It uses two vectors as a starting and an end point, it then subtracts them from each other to find the length of the resulting vector. The code is straight forward and simple. It uses a vector class provided to us as a blueprint to perform its operations. The distance between two points formula which is, $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$, is used to obtain this calculation. In detail this formula subtracts each x, y and z co-ordinate from each other while squaring each of the three results and then square rooting them. In code form it looks like this,

```
void Vector::distancePoints(Vector &vector2)
{
    cout << (pow(pow((x - vector2.x), 2) + pow((y - vector2.y), 2) + pow((z - vector2.z), 2), 0.5)) << endl << endl;
}
```

The class looks like this,

```

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);
    void distancePoints(Vector &vector2);
    float x;
    float y;
    float z;
};

```

Test 1

Description: In this test the values for vector one are x=5, y=3 and z=1. The values for vector two are x=17, y=4 and z=8.

Expected Result: 13.92

Actual Result: 13.92

Pass/Fail: Pass

```

C:\Users\Dan\Documents\Uni Year2\Assig
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 1

13.9284

1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?:

```

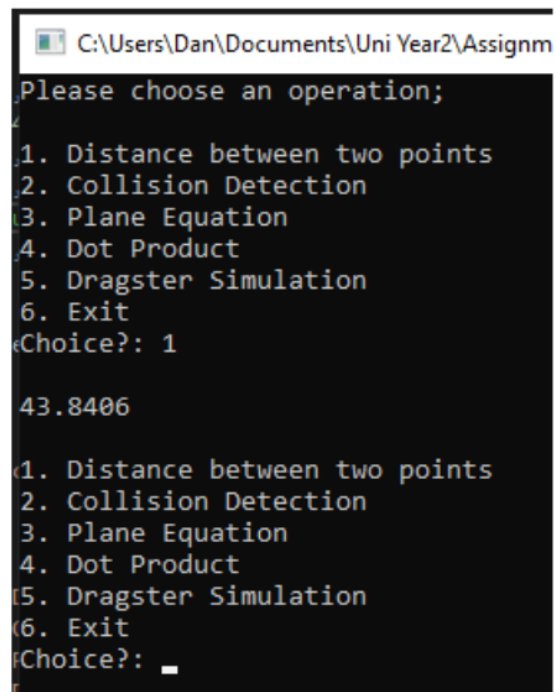
Test 2

Description: In this test the values for vector one are $x=5$, $y=20$ and $z=7$. The values for vector two are $x=44$, $y=2$ and $z=6$.

Expected Result: 43.84

Actual Result: 43.84

Pass/Fail: Pass



```
C:\Users\Dan\Documents\Uni Year2\Assignm
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 1

43.8406

1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: _
```

Test 3

Description: In this test the values for vector one are $x=10$, $y=8$ and $z=9$. The values for vector two are $x=22$, $y=17$ and $z=54$.

Expected Result: 47.43

Actual Result: 47.43

Pass/Fail: Pass

C:\Users\Dan\Documents\Uni Year2\Assig

Please choose an operation;

1. Distance between two points
 2. Collision Detection
 3. Plane Equation
 4. Dot Product
 5. Dragster Simulation
 6. Exit
- Choice?: 1

47.4342

1. Distance between two points
 2. Collision Detection
 3. Plane Equation
 4. Dot Product
 5. Dragster Simulation
 6. Exit
- Choice?:

$$\begin{array}{l} \begin{array}{l} 17, 4, 1 \\ 5, 3, 8 \end{array} \\ \sqrt{(5-17)^2 + (3-4)^2 + (8-1)^2} \\ \sqrt{-144 + -1 + 49} \\ \sqrt{-96} \\ \\ \begin{array}{l} 17, 4, 1 \\ 5, 3, 8 \end{array} \\ \sqrt{(17-5)^2 + (4-3)^2 + (8-1)^2} \\ \sqrt{144 + 1 + 49} \\ \sqrt{194} \\ \\ \begin{array}{l} 5, 20, 87 \\ 44, 2, 6 \end{array} \\ \sqrt{(44-5)^2 + (2-20)^2 + (6-87)^2} \\ \sqrt{(39)^2 + (-18)^2 + (-1)^2} \\ \sqrt{1521 + (-324) + (-1)} \\ \sqrt{1,196} \\ \\ \begin{array}{l} 10, 8, 9 \\ 22, 17, 54 \end{array} \\ \sqrt{(22-10)^2 + (17-8)^2 + (54-9)^2} \\ \sqrt{(12)^2 + (9)^2 + (45)^2} \\ \sqrt{144 + 81 + 2,025} \\ \sqrt{2,250} \end{array}$$

Planes

"A plane in 3D space can be thought of as a flat surface extending indefinitely in all directions. It can be described in several different ways. For example, by: Three points not on a straight line (forming a triangle on the plane) A normal and a point on the plane A normal and a distance from the origin." [3]. This method allows the distance between the plane and a point in 3D space to be calculated. To do this the code gets the length of vector N by using the distance between two points formula, $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$. The length is then normalized by dividing each co-ordinate by the length which was found earlier. Once this is done the dot product for normalized N and Vector X is found, as well as the negative dot product for normalized N and point P. The two dot products are added together, by using the prefix abs we can make sure that the result is always a positive or absolute number. The same class that was created can also calculate the dot product as a standalone result. The code looks like this,

```
void Plane::planeequation()
{
    float length = (pow(pow((n.x), 2) + pow((n.y), 2) + pow((n.z), 2), 0.5));

    Vector normal;
    Vector x(1, 2, 3);
    normal.x = (n.x / length);
    normal.y = (n.y / length);
    normal.z = (n.z / length);

    float dotproduct1 = -(normal.x * p.x) + (normal.y * p.y) + (normal.z * p.z);

    float dotproduct2 = (x.x * normal.x) + (x.y * normal.y) + (x.z * normal.z);

    float total = abs(dotproduct2 + dotproduct1);

    cout << total << endl << endl;
}

void Plane::dotproduct()
{
    float product = (p.x * n.x) + (p.y * n.y) + (p.z * n.z);

    cout << product << endl << endl;
}

Plane::Plane(int a, int b, int c, int d, int e, int f)
{
    p.x = a;
    p.y = b;
    p.z = c;
    n.x = d;
    n.y = e;
    n.z = f;
}
```

The class looks like this,

```
class Plane : Vector
{
public:
    void planeequation();
    void dotproduct();
    Vector n;
    Vector p;
    Plane(int a, int b, int c, int d, int e, int f);
};
```

Plane equation test 1

Description: In this test the values for P are x=3, y=2, z=2, X is x=1, y=2, z=3 and N is x=1, y=2, z=3.

Expected Result: 7.16

Actual Result: 7.21

Pass/Fail: Fail

Handwritten calculations for the plane equation test:

$$\begin{aligned} P &= 3 \ 2 \ 2 & \text{length is } 3.74 \\ N &= 1 \ 2 \ 3 \\ X &= 1 \ 2 \ 3 \end{aligned}$$
$$\begin{aligned} \text{Normal } x &= \left(\frac{1}{3.74} \right) \\ \text{Normal } y &= \left(\frac{2}{3.74} \right) \\ \text{Normal } z &= \left(\frac{3}{3.74} \right) \end{aligned}$$
$$\begin{aligned} \text{Normal } x &= (0.26) \\ \text{Normal } y &= (0.53) \\ \text{Normal } z &= (0.80) \end{aligned}$$
$$\begin{aligned} &(0.26 \times 3) + (0.53 \times 2) + (0.80 \times 2) \\ &(0.78) + (1.06) + (1.60) \\ &= 3.44 \end{aligned}$$
$$\begin{aligned} &(1 \times 0.26) + (2 \times 0.53) + (3 \times 0.80) \\ &0.26 + 1.06 + 2.4 \\ &= 3.72 \end{aligned}$$
$$\text{Total} = 3.72 + 3.44 = 7.16$$

Plane equation test 2

Description: In this test the values for P are $x=5, y=7, z=1$, X is $x=6, y=9, z=4$ and N is $x=2, y=2, z=3$.

Expected Result: 16.56

Actual Result: 16.73

Pass/Fail: Fail

Handwritten calculations for Plane equation test 2:

$$\text{length} = \sqrt{2^2 + 2^2 + 3^2}$$
$$\sqrt{4+4+9}$$
$$\sqrt{17} = 4.12$$
$$\text{Normal } x = \left(\frac{2}{4.12} \right)$$
$$y = \left(\frac{2}{4.12} \right)$$
$$z = \left(\frac{3}{4.12} \right)$$
$$x \quad y \quad z$$
$$0.48 \quad 0.48 \quad 0.72$$
$$(0.48 \times 5) + (0.48 \times 7) + (0.72 \times 1)$$
$$2.4 + 3.36 + 0.72$$
$$DP1 = 6.48$$
$$(0.48 \times 6) + (0.48 \times 9) + (0.72 \times 4)$$
$$2.88 + 4.32 + 2.88$$
$$DP2 = 10.08$$
$$\text{Total} = (10.08 + 6.48) = 16.56$$

Plane equation test 3

Description: In this test the values for P are $x=7, y=11, z=4$, X is $x=15, y=1, z=2$ and N is $x=6, y=5, z=6$.

Expected Result: 22.8

Actual Result: 23.04

Pass/Fail: Fail

length is
9.84

normal $x = 6 / 9.84$
 normal $y = 5 / 9.84$
 normal $z = 6 / 9.84$

nor x nor y nor z
 0.60 0.50 0.60

$(0.60 \times 7) + (0.50 \times 11) + (0.60 \times 4)$
 4.2 + 5.5 + 2.4
 DP1 = 12.1

$(0.60 \times 15) + (0.50 \times 1) + (0.60 \times 2)$
 9 + 0.50 + 1.2
 DP2 = 10.7

Total = 22.8

Dot product test 1

Description: In this test the values for P are $x=4, y=7, z=2$, N is $x=9, y=1, z=3$.

Expected Result: 49

Actual Result: 49

Pass/Fail: Pass

```
C:\Users\Dan\Documents\Uni Year2\Assignm
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 4

Dot product is: 49

1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: _
```

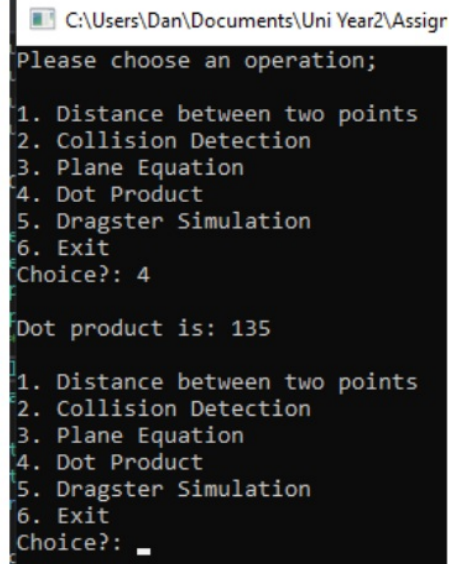
Dot product test 2

Description: In this test the values for P are $x=1, y=9, z=14$, N is $x=33, y=2, z=6$.

Expected Result: 135

Actual Result: 135

Pass/Fail: Pass



```
C:\Users\Dan\Documents\Uni Year2\Assignm...
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 4

Dot product is: 135
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?:
```

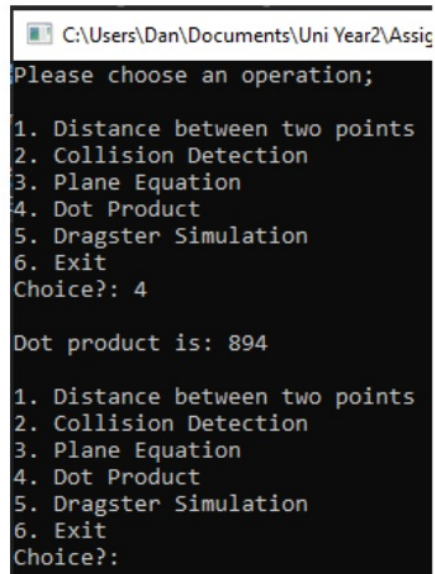
Dot product test 3

Description: In this test the values for P are $x=67, y=113, z=22$, N is $x=5, y=3, z=10$.

Expected Result: 894

Actual Result: 894

Pass/Fail: Pass



```
C:\Users\Dan\Documents\Uni Year2\Assignm...
Please choose an operation;
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?: 4

Dot product is: 894
1. Distance between two points
2. Collision Detection
3. Plane Equation
4. Dot Product
5. Dragster Simulation
6. Exit
Choice?:
```

Bibliography

[1] Brilliant, "Euler's Method", February 4, 2020. Accessed on February 4, 2020. [Online]. Available: <https://brilliant.org/wiki/eulers-method/>

[2] James M. Van Verth and Lars M. Bishop, Essential Mathematics for Games and Interactive Applications A Programmer's Guide, 2nd ed. Burlington, USA: Morgan Kaufmann Publishers, 2008, p.616

[3] Christer Ericson, Real-Time Collision Detection. San Francisco, USA: Morgan Kaufmann Publishers, 2005, p.20, p.54

GAM Assignment Daniel O'Dwyer 1801164

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

64/100

PAGE 1

PAGE 2

PAGE 3



Comment 1

right of what???

Avoid over-reliance on blocks of quoted text.



Comment 2

no comment



Comment 3

backwards Euler

PAGE 4



Comment 4

in what way, how?

PAGE 5

PAGE 6

PAGE 7

Comment 5



accel?



Comment 6

big mismatch between the two herre Explore why this is so.

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

GRADING FORM: SMU GENERIC_GJ

DANIEL
O'DWYER

64

RESEARCH-INFORMED LITERATURE

 Some good use of refs. Key areas identified and ref'd. **7**

KNOWLEDGE AND UNDERSTANDING

 Essential topics discussed. Good. Some issues in translating integration methods to code but you did try both Verlet and Euler so well done. Shame you couldn't identify accuracy issue in one of these. **13.5**

ANALYSIS

 Adv. and disadvantages identified and discussed. Could be more detailed here. Results for Verlet and Euler provided and some attempts made at reconciling different results. **13**

PRACTICAL APPLICATION AND DEP.

 Avoid outputs in methods that just calculate a value. Comments light. Some issues in Vertlet method. Output results to file please. Plane equation almost right - check sign. **27**

SKILLS FOR PROFESSIONAL PRAC.

 Some good refs here - well done. Structured report. Good. **3.5**

