

Bartosz_Galantowicz_1700994_ ACCA5017_Games_Architecture _&_Mechanics_Assignment

by Bartosz Galantowicz

FILE	2275_BARTOSZ_GALANTOWICZ_BARTOSZ_GALANTOWICZ_1700994_AC CA5017_GAMES_ARCHITECTURE__MECHANICS_ASSIGNMENT_234537_ 472364374.DOCX (1.3M)		
TIME SUBMITTED	07-FEB-2020 10:13PM (UTC+0000)	WORD COUNT	3441
SUBMISSION ID	119778467	CHARACTER COUNT	18391

Bartosz Galantowicz

1700994

Part 1a: Games Physics

Games Architecture and Mechanics

ACCA5017

Gordon Dickers

Table of Contents

Abstract.....	3
Introduction	3
Design.....	3
Task 1	4
Task 2	14
Task 3	25
Summary.....	31

Abstract

The report is evidence of the knowledge in the field of games mechanics and architecture that has been cultivated over the second-year academic studies at the University of Wales Trinity Saint David. Furthermore, the main purpose of the report is to implement a program for solving given examples related to the application of physics in programming.

Introduction

C++ is a high-level programming language, in which the assigned program is coded. Moreover, the IDE is Visual Studio 2017, and the software is a Windows Desktop application. The given tasks are based on physics issues, which must be solved using the knowledge and skills obtained during the study of the module. For each task, the theoretical background is discussed. Accordingly, an example of using a formula in practice is shown and tested.

Design

The program design idea is to connect all tasks in one menu, which is displayed to a user. Hence, a specific task may be chosen by a user to perform a simulation and tests. What is more, it is possible to repeat a function until it is not decided to exit the program.

```
39 int main(void)
40 {
41     // Store user choice
42     unsigned int userChoice = 0;
43
44     // Display the menu of the program in the console windows
45     do
46     {
47         cout << "\n" << "*****"
48         cout << "Welcome in assignment menu!" << endl;
49         cout << "Choose a task to perform:" << endl;
50         cout << "1. Task 1" << endl;
51         cout << "2. Task 2" << endl;
52         cout << "3. Task 3" << endl;
53         cout << "4. Exit the application." << endl;
54
55         // Get user input
56         userChoice = readIntValue(1, 4, "");
57         // Proceed with a choosen operation
58         proceesOperation(userChoice);
59     } while (userChoice != 4);
60
61
62
63     return 0;
64 }
```

Fig.1 The picture of the designed menu in C++.

The algorithms for solving the tasks are coded in C++ language in Visual Studio 2017 IDE. Furthermore, the principles of object-oriented programming and code of practice for programmers have been kept in mind during work on the project.

Task 1

The given task aims are to perform a simulation of the movement of a spring-powered dragster car in a computer program. A tractive force and a drag force are applied to the model. Hence they are included in the calculation. However, the main task of the coded algorithm is to determine the velocity, position and acceleration for each step. The experiment is carried out from the starting point of motion simulation until a halt.

Numerical integrations methods are implemented to describe the motion of the car model. These approaches are used to find out an approximation of values, due to getting an awareness of complication of an analysed example. Furthermore, some functions cannot be analytically integrated, or the efficiency of the completed calculations is low and is worth and sufficiently to work with estimated data.

To solve the given assignment, a Euler's Method was implemented, because basic Newtonian physics is not appropriate for solving problems, including the tendency to change (non-constant) values [1] [2]. For example, in the simulation of the motion of the car model, the acceleration and speed are affected by drag and tractive forces. Accordingly, an occurrence of shifts is done over the simulation period.

Starting the calculation is dependent on the initial value problem of the form:

$$\begin{aligned}y' &= f(x, y) \\ y(x_0) &= y_0\end{aligned}$$

And decision upon what interval (a rate of change) is required to accurate the precision of the provided solution. After defining a length of the step, it is possible to find out the solution for the rest of the coordinates by using the iterative formulas:

$$\begin{aligned}x_{n+1} &= x_n + h \\ y_{n+1} &= y_n + h \times f(x_n, y_n)\end{aligned}$$

Nevertheless, it is important to be aware of the inaccuracy of using a numerical method to solve a problem. The accuracy of outcomes is based on the size of the step. Namely, the scale of errors per step for approximations, in comparison with the original values, is manipulated

by the length (h). What is more, the estimated values are less appropriate as farther their coordinates are in relation to a known initial coordinate [1] [2].

The purpose of Euler's Method is to perform complicated calculations and solve complexed derivatives to obtain some approximated vision of a rate of changes. Although it is an inaccurate approach, especially when solving by hand, the application of it is used in the fields of computing and science. Solving more complexed and larger problems is more efficient and precise due to the computational power of modern computers. What is more is still worth to find out the estimation rather than an exact result, because in some cases, a required outcome is fulfilling by applying rough values and it is no need to overload performance of the computing machine. For instance, the evaluation of the effects and performance are an important part of games development, where physics is a base for the virtually created world [3].

The assumption of the created program is that a car is an object of its class, and the right simulation is performed in the main function, which is also responsible for calculating environmental aspects of the process, such as gravity or air density.

```
1  #pragma once
2
3  #define MASS 1 // Mass of the vechicle
4  #define VECHICLE_FRONTAL_AREA 0.01
5
6  class cCar
7  {
8  public:
9      cCar(); // Constructor of the class, it sets all private values to zero
10     ~cCar();
11     void setAcceleration(float *); // Use the formula to calculate acceleration
12     float getAcceleration(); // Return acceleration
13     void setVelocity(float *); // Use the formula to calculate velocity
14     float getVelocity(); // Return velocity
15     void setPos(float *); // Use the formula to calculate position
16     float getPos(); // Return acceleration
17
18 private:
19     float velocity;
20     float acceleration;
21     float posX;
22
23     };
```

Fig.2 The screenshot of the header of cCar class, in which attributes of the model are held and also the related values are calculated and stored, such as velocity, acceleration or position.

```

10 void cCar::setAcceleration(float *sumForces)
11 {
12     acceleration = *sumForces / MASS;
13 }

```

```

20 void cCar::setVelocity(float *timeStep)
21 {
22     velocity = velocity + (*timeStep * acceleration);
23 }

```

```

36 void cCar::setPos(float *timeStep)
37 {
38     posX = posX + (*timeStep * velocity);
39 }

```

Fig.3 The figures are showing some of the functions of the cCar class, which are responsible for the calculation of velocity, acceleration and position.

```

9  #define ROLLING_RESISTANCE_COEFFICIENT 0.21
10 #define GRAVITY 9.81
11 #define DRAG_COEFFICIENT 0.34
12 #define AIR_DENSITY 1.2
13
14
15 using namespace std;
16 float stepTime = 0.06;
17
18 float getDragForce(float mass, float frontalArea, float velocity);
19 float getTractiveForce(float distance);
20 float sumForces(float dragForce, float tractiveForce);

```

Fig.4 The values for specific environment variables are defined in the main file.

```

// Create an excel document to save collected data
ofstream calculationSaver("Task 1 Calculation.csv", ios::out | ios::trunc | ios::in);

```

Fig.5 Due to using fstream library, the data is saved as an excel file in the main folder of the program.

```

do // Do until the movement is finished
{
    totalForce = sumForces(getDragForce(MASS, VEHICLE_FRONTAL_AREA, car->getVelocity()), getTractiveForce(car->getPos()));
    car->setAcceleration(&totalForce);
    car->setVelocity(&stepTime);
    car->setPos(&stepTime);

    // Display the result in the console window
    cout << fixed << setprecision(7) << stepCounter + 1 << ", "
        << time << ", " << car->getAcceleration() << ", " << car->getVelocity() << ", " << car->getPos() << endl;

    // Save the result as an excel document
    calculationSaver << fixed << setprecision(7) << stepCounter + 1 << ", "
        << time << ", " << car->getAcceleration() << ", " << car->getVelocity() << ", " << car->getPos() << endl;

    // Update time
    time = time + stepTime;

    stepCounter++;
} while (car->getVelocity() > 0);

```

Fig.6 The example of the loop, which is responsible for the calculation of the simulation until the car stops.

```

138 // Get a drag force
139 float getDragForce(float mass, float frontalArea, float velocity)
140 {
141     if (velocity < 0)
142     {
143         return 0;
144     }
145     else
146     {
147         return ((ROLLING_RESISTANCE_COEFFICIENT * mass * GRAVITY) + ((0.5 * (DRAG_COEFFICIENT * AIR_DENSITY * frontalArea) * (pow(velocity, 2)))));
148     }
149 }
150
151 // Get a tractive force
152 float getTractiveForce(float distance)
153 {
154     if (distance <= 0)
155     {
156         return 1.5*(16 - distance);
157     }
158     else
159     {
160         return 0;
161     }
162 }

```

Fig.7 The drag force and tractive force are found by the functions presented in the picture above.

Step	Time	Acceleration	Velocity	Position
1	0	21.9398994	1.316394	0.0789836
2	0.06	21.8178902	2.6254673	0.2365117
3	0.12	21.5710716	3.9197316	0.4716955
4	0.18	21.2010117	5.1917925	0.7832031
5	0.24	20.7101078	6.4343987	1.1692671
6	0.3	20.1015396	7.640491	1.6276965
7	0.36	19.3792648	8.8032465	2.1558914
8	0.42	18.5479679	9.9161243	2.7508588
9	0.48	17.6130199	10.9729052	3.4092331
10	0.54	16.5804253	11.9677305	4.1272969
11	0.6	15.4567728	12.8951368	4.9010053
12	0.66	14.2491722	13.7500868	5.7260103
13	0.72	12.9651918	14.527998	6.5976901
14	0.78	11.6127958	15.2247658	7.5111761
15	0.84	10.2002773	15.8367825	8.4613829
16	0.9	-2.5717394	15.682478	9.4023314
17	0.96	-2.5618179	15.5287685	10.3340578
18	1.02	-2.552031	15.3756466	11.2565966
19	1.0799999	-2.5423775	15.2231035	12.1699829
20	1.1399999	-2.5328555	15.0711327	13.0742512
21	1.1999998	-2.5234637	14.9197245	13.9694347
22	1.2599998	-2.5142002	14.7688723	14.855567
23	1.3199997	-2.505064	14.6185684	15.7326813
24	1.3799996	-2.4960532	14.4688053	16.6008091
25	1.4399996	-2.4871664	14.3195753	17.4599838
26	1.4999995	-2.4784024	14.1708708	18.310236
27	1.5599995	-2.4697597	14.0226851	19.151598
28	1.6199994	-2.4612367	13.8750105	19.9840984
29	1.6799994	-2.4528325	13.7278404	20.8077698
30	1.7399993	-2.4445453	13.5811682	21.6226406
31	1.7999992	-2.4363742	13.4349861	22.4287395
32	1.8599992	-2.4283175	13.2892866	23.2260971
33	1.9199991	-2.4203744	13.1440639	24.01474
34	1.9799991	-2.4125435	12.9993114	24.7946987

35	2.039999	-2.4048235	12.8550224	25.566
36	2.099999	-2.3972132	12.7111893	26.3286705
37	2.1599989	-2.3897116	12.5678062	27.0827389
38	2.2199988	-2.3823175	12.4248676	27.8282318
39	2.2799988	-2.3750298	12.2823658	28.5651741
40	2.3399987	-2.3678472	12.140295	29.2935925
41	2.3999987	-2.360769	11.9986486	30.0135117
42	2.4599986	-2.3537939	11.8574209	30.7249565
43	2.5199986	-2.3469207	11.7166061	31.4279537
44	2.5799985	-2.3401489	11.5761976	32.1225243
45	2.6399984	-2.333477	11.4361887	32.8086967
46	2.6999984	-2.3269043	11.2965746	33.4864922
47	2.7599983	-2.3204298	11.1573486	34.1559334
48	2.8199983	-2.3140523	11.0185051	34.8170433
49	2.8799982	-2.3077712	10.8800392	35.4698448
50	2.9399981	-2.3015854	10.7419443	36.1143608
51	2.9999981	-2.2954943	10.6042147	36.7506142
52	3.059998	-2.2894967	10.4668446	37.378624
53	3.119998	-2.2835917	10.3298292	37.9984131
54	3.1799979	-2.2777789	10.1931629	38.6100044
55	3.2399979	-2.2720571	10.0568399	39.2134132
56	3.2999978	-2.2664256	9.9208546	39.8086662
57	3.3599977	-2.2608836	9.785202	40.3957787
58	3.4199977	-2.2554305	9.6498766	40.9747696
59	3.4799976	-2.2500651	9.5148726	41.5456619
60	3.5399976	-2.244787	9.3801851	42.1084747
61	3.5999975	-2.2395952	9.2458096	42.6632233
62	3.6599975	-2.2344894	9.1117401	43.2099266
63	3.7199974	-2.2294686	8.977972	43.7486038
64	3.7799973	-2.2245321	8.8445005	44.279274
65	3.8399973	-2.2196794	8.7113199	44.8019524
66	3.8999972	-2.2149096	8.5784254	45.316658
67	3.9599972	-2.2102222	8.4458122	45.8234062
68	4.0199971	-2.2056167	8.3134756	46.322216
69	4.0799971	-2.2010922	8.1814098	46.8130989
70	4.139997	-2.1966484	8.0496111	47.2960739

71	4.1999969	-2.1922843	7.9180741	47.7711601
72	4.2599969	-2.1879997	7.7867942	48.238369
73	4.3199968	-2.1837938	7.6557665	48.6977158
74	4.3799968	-2.179666	7.5249867	49.1492157
75	4.4399967	-2.1756158	7.3944497	49.5928841
76	4.4999967	-2.1716428	7.2641511	50.0287323
77	4.5599966	-2.1677465	7.1340861	50.4567757
78	4.6199965	-2.1639261	7.0042505	50.8770294
79	4.6799965	-2.1601815	6.8746395	51.2895088
80	4.7399964	-2.1565118	6.7452488	51.6942253
81	4.7999964	-2.1529167	6.6160736	52.0911903
82	4.8599963	-2.1493957	6.4871097	52.4804153
83	4.9199963	-2.1459484	6.3583527	52.8619156
84	4.9799962	-2.1425745	6.2297983	53.2357025
85	5.0399961	-2.1392732	6.1014419	53.6017876
86	5.0999961	-2.1360443	5.973279	53.960186
87	5.159996	-2.1328874	5.8453059	54.3109055
88	5.219996	-2.129802	5.7175179	54.6539574
89	5.2799959	-2.1267877	5.5899105	54.9893532
90	5.3399959	-2.1238441	5.4624801	55.3171005
91	5.3999958	-2.120971	5.3352218	55.6372147
92	5.4599957	-2.1181679	5.2081318	55.9497032
93	5.5199957	-2.1154342	5.0812058	56.2545738
94	5.5799956	-2.1127701	4.9544396	56.5518417
95	5.6399956	-2.1101749	4.8278294	56.8415108
96	5.6999955	-2.1076481	4.7013702	57.1235924
97	5.7599955	-2.1051898	4.5750589	57.3980942
98	5.8199954	-2.1027997	4.4488912	57.6650276
99	5.8799953	-2.100477	4.3228626	57.9244003
100	5.9399953	-2.0982218	4.1969695	58.1762199
101	5.9999952	-2.0960336	4.0712075	58.4204941
102	6.0599952	-2.0939124	3.9455729	58.6572266
103	6.1199951	-2.0918579	3.8200614	58.8864288
104	6.1799951	-2.0898695	3.6946692	59.1081085
105	6.239995	-2.0879471	3.5693924	59.3222733
106	6.2999949	-2.0860908	3.444227	59.5289268

107	6.3599949	-2.0842998	3.319169	59.7280769
108	6.4199948	-2.0825744	3.1942146	59.9197311
109	6.4799948	-2.080914	3.0693598	60.1038933
110	6.5399947	-2.0793188	2.9446006	60.280571
111	6.5999947	-2.0777881	2.8199334	60.4497681
112	6.6599946	-2.0763221	2.695354	60.6114883
113	6.7199945	-2.0749204	2.5708587	60.7657394
114	6.7799945	-2.0735829	2.4464438	60.9125252
115	6.8399944	-2.0723095	2.3221052	61.0518532
116	6.8999944	-2.0711	2.1978393	61.1837234
117	6.9599943	-2.0699542	2.073642	61.3081436
118	7.0199943	-2.068872	1.9495097	61.4251137
119	7.0799942	-2.0678532	1.8254385	61.5346413
120	7.1399941	-2.0668976	1.7014246	61.6367264
121	7.1999941	-2.0660055	1.5774642	61.7313728
122	7.259994	-2.0651762	1.4535537	61.8185844
123	7.319994	-2.0644102	1.329689	61.898365
124	7.3799939	-2.0637069	1.2058666	61.9707184
125	7.4399939	-2.0630665	1.0820826	62.0356445
126	7.4999938	-2.0624886	0.9583333	62.0931435
127	7.5599937	-2.0619736	0.8346149	62.143219
128	7.6199937	-2.0615211	0.7109236	62.1858749
129	7.6799936	-2.061131	0.5872558	62.2211113
130	7.7399936	-2.0608037	0.4636075	62.2489281
131	7.7999935	-2.0605385	0.3399752	62.2693253
132	7.8599935	-2.0603359	0.2163551	62.2823067
133	7.9199934	-2.0601954	0.0927434	62.2878723
134	7.9799933	0	0	62.2860222

Fig.8 The table is showing the result of the simulation of the movement of a spring-powered model dragster.

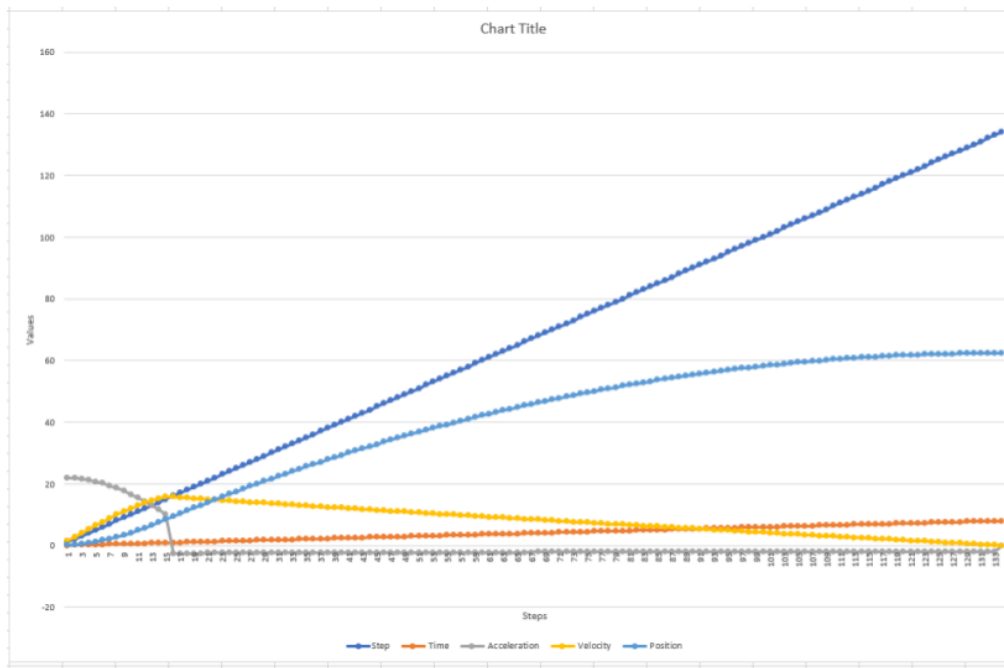


Fig.9 The result of the simulation has been plotted into the chart, which is indicating the values for step, time, acceleration, velocity and position over the simulation period.

Physics in games are implemented to reflect realism. However, some decisions must be made to determine when and where various physic formulas can be applied by games developers, due to maintaining good proportions between quality and performance.

Previously discussed explicit Euler method is one of the tools used by programmers to get a conceptual understanding from other integration techniques. This solution is implemented when the impact of error values is not noticeable in gameplay and the physic action performed in a game is short, e.g. a jump of character [3] [4] [5].

$$v_{n+1} = v_n + (a_n \times dt)$$

$$p_{n+1} = p_n + (v_n \times dt)$$

Fig.10 The equations for calculating the velocity and then position by referring to the explicit Euler method.

From the other hand, there is a Backwards Euler technique of calculating one derivative allowing to get the estimation of the previous example. In comparison with explicit Euler approach, this is more stable and accurate but slower in computation. Also, it is more complex

as the number of kept variables is increased although disadvantages Backwards Euler is used to manipulating on friction, drag and changes in acceleration in games [3] [4] [5].

$$\begin{aligned}v_{n+1} &= V_n + (a_{n+1} \times dt) \\p_{n+1} &= p_n + (v_{n+1} \times dt)\end{aligned}$$

Fig.11 The equations for calculating the velocity and then position by referring to the Backward Euler method.

Moreover, Semi-Implicit Euler is the solution, which is a connection of already introduced techniques (explicit Euler, Backward Euler). Therefore, the velocity is calculated using the explicit Euler equation and finding the position is figured out by application of the Backward Euler's method. The advantages of both approaches are the effect of this symbiosis. Namely, the results are more precise, and the implementation is much faster. This method is mainly used to compute a rigid body system [3] [4] [5].

$$\begin{aligned}v_{n+1} &= v_n + (a_n \times dt) \\p_{n+1} &= p_n + (v_{n+1} \times dt)\end{aligned}$$

Fig.12 The equations for calculating the velocity and then position by referring to the Semi-Implicit Euler's method.

The other idea of creation of physics in games is Verlet Integration. This is more accurate than Euler's techniques. Also, it is still simply to implementation and, the time of performance is reduced. Additionally, the position is calculated from acceleration (not from velocity such as Euler's methods), and then the velocity is calculating from the position. However, the disadvantage is a low level of precision while the step value is not constant. In the games industry, it is employed to simulate projectile trajectories, particles system or cloth effects [3] [4] [5].

$$\begin{aligned}p_{n+1} &= 2 \times p_n - p_{n-1} + a \times dt^2 \\v_{n+1} &= (p_n - p_{n-1}) / dt\end{aligned}$$

Fig.13 The equations for calculating the position and then velocity by referring to the Verlet Integration method.

Because Verlet Integration is not accurate enough for approximating velocity, the other solution has been invented. The precision of velocity is obtained due to the Velocity Verlet method, and the process of finding a result is split into four stages. Therefore, the velocity is calculated twice. Firstly, at a half step and another time after computation of the next acceleration. Nevertheless, lost of some efficiency is compared to the capability of

implementation of rapid changes in a velocity, which is an important element of many games e.g. racing games [3] [4] [5].

$$\begin{aligned}v_{n+1/2} &= v_n + a_n \times dt/2 \\p_{n+1} &= p_n + v_{n+1/2} \times dt \\a_{n+1} &= F/m \\v_{n+1} &= v_{n+1/2} + a_{n+1} \times dt / 2\end{aligned}$$

Fig.14 The equations for calculating the velocity and then position by referring to the Velocity Verlet method.

Summarise, there are many types of integration methods which have been developed to recreate physics in games. Advantages and disadvantages of each of mentioned methods are to the disposal of a game programmer whose role is to decide when a flow performance of the production is essential and when is a possibility to design something, demanding a huge amount of computational power but, with the highest quality effects to make players enjoy.

Task 2

The second task of the assignment is focused on operations on vectors. Namely, it is required to perform the series of test on the given vector class with the implementation of mathematical theory. What is more, this theory is investigated and described in this report.

Vectors are defined by two factors: direction, which is graphically represented by a pointing arrow, and magnitude – a length. The examples of vectors in everyday life scenario are, the velocity – its magnitude is described by speed – and force – its length is indicated by strength. Also, the directions are dependent on the given examples. However, the manipulation over vectors is possible, due to the cartesian coordinates system as it is allowed to determinate a position – a point of reference for vector [6] [7].



Fig.15 The structure of a vector (https://mathinsight.org/vector_introduction).

In the assumption that the coordinate system is combined with three-dimensional space defined by X, Y, Z-axis it is possible to position a thing in space knowing their exact coordinates, e.g. point Q is at position (x, y, z). Finding the position of a vector is begun by translation of the vector to the origin of the space and applying its magnitude following the direction located at some position. Thereby, the extra calculations, such as multiplications by scalar, addition or subtraction, can be carried out on that vector [6] [7].

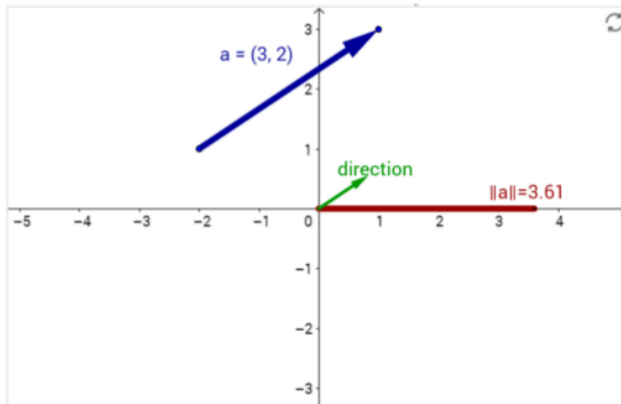


Fig.16 Allocating a vector in a coordinate system
(https://mathinsight.org/vectors_cartesian_coordinates_2d_3d).

A small letter with an arrow above or a bold letter is a valid notation for vectors.

→

a

OR

a

The very first operation on vector is to find its magnitude. In the coordinate system, it is done by applying Pythagoras theorem regarding the equation:

$$|p| = \sqrt{x^2 + y^2 + z^2}$$

Thereby, the distance between to points is calculated by the equation:

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

```
1  #pragma once
2  /***(note x,y,z are not private-should they be?)
3  #include <math.h>
4  #ifndef VECTOR_H
5  #define VECTOR_H
6
7  class cVector
8  {
9  public:
10
11     //construct
12     cVector(float x, float y, float z);
13     cVector(void);
14     void Scale(float s);
15     void Add(cVector & v1);
16     void Add(cVector & v1, cVector & v2);
17
18     float getMagnitude();
19     float getDistanceBetweenTwoPoints(cVector & v2);
20     cVector getVector(void);
21     cVector getCrossProduct(cVector & v1, cVector & v2);
22     float getValueXYZByIndex(int i);
23     float getDotProduct(cVector & n, cVector & p0, cVector & p1);
24     cVector formVector(cVector & point1, cVector & point2);
25
26 private://these are currently public - should they be private?
27     float x;
28     float y;
29     float z;
30 };
31 #endif
32
```

Fig.17 The header of the vector class. The access to x,y and z coordinates has been solved by implementation of a specific function (“get-type” function).

```

33 // Calculate the magnitude
34 float cVector::getMagnitude()
35 {
36     return fabsf(sqrt((pow(x,2)) + (pow(y,2)) + (pow(z,2))));
37 }
38
39 // Calculate the distance between two points
40 float cVector::getDistanceBetweenTwoPoints(cVector & v2)
41 {
42     return fabsf(sqrt((pow(v2.x - x, 2)) + (pow(v2.y - y, 2)) + (pow(v2.z - z, 2))));
43 }
44
45 // Get a singular value for coordinate by using index
46 float cVector::getValueXYZByIndex(int i)
47 {
48     float coordinates[3] = { x, y, z };
49
50     return coordinates[i];
51 }
52
53 // Calculate the cross product of two vectors
54 cVector cVector::getCrossProduct(cVector & v1, cVector & v2)
55 {
56     cVector crossProduct;
57     crossProduct.x = (v1.y * v2.z) - (v1.z * v2.y);
58     crossProduct.y = (v1.x * v2.z) - (v1.z * v2.x);
59     crossProduct.z = (v1.x * v2.y) - (v1.y * v2.x);
60
61     return crossProduct;
62 }
63
64 // Calculate the dot product of two vectors
65 float cVector::getDotProduct(cVector & n, cVector & p0, cVector & p1)
66 {
67     float dotProduct = ((n.x * (p1.x - p0.x)) + (n.y * (p1.y - p0.y)) + (n.z * (p1.z - p0.z)));
68
69     return dotProduct;
70 }

```

```

72 // Form vector from two points
73 cVector cVector::formVector(cVector & point1, cVector & point2)
74 {
75     return cVector(point2.x - point1.x, point2.y - point1.y, point2.z - point1.z);
76 }

```

Fig.18 Some of the functions have been already provided by supervision. However, the screenshot is presenting additional functions created for task 2.

Attempt	Test number	Description	Expected Result	Actual Result	Pass/Fail
1	A = (3, 2, 0) B = (-3, 0, 0)	Finding the distance between two points.	6.324555	6.32456	Pass

2	A = (-7, 6, -3) B = (-7, 6, -3)	Finding the distance between two points.	0	0	Pass
3	A = (-2.4, -0.2, 3.7) B = (-11, 0.8, 6.6)	Finding the distance between two points.	9.130717	9.13072	Pass

```

Provide coordinates for the first point:
x: 3
y: 2
z: 0
Provide coordinates for the second point:
x: -3
y: 0
z: 0
Distance between given points is 6.32456

P0 is the point set on the plane.
P0 = (1.000,-2.000,0.000)

```

```

Provide coordinates for the first point:
x: -7
y: 6
z: -3
Provide coordinates for the second point:
x: -7
y: 6
z: -3
Distance between given points is 0

```

```

Welcome in assignment menu!
Choose a task to perform:
1. Task 1
2. Task 2
3. Task 3
4. Exit the application.
2
Provide coordinates for the first point:
x: -2.4
y: -0.2
z: 3.7
Provide coordinates for the second point:
x: -11
y: 0.8
z: 6.6
Distance between given points is 9.13072

```

Fig.19 The tests and their results.

The second assigned work for task two is to implement a plane class, including its attributes – a vector describing the normalised plane normal and a point, which is the point of the plane.

A plane is a two-dimensional, flat surface that length and width are infinite. Additionally, it can be presented in a cartesian coordinates system. Due to using vectors, it is manageable to find out the normal to the surface of the plane, which is resulted in a normal vector perpendicular to the surface. The normal vector is the outcome of the cross product of a known point and any other point set on the surface [8] [9].

Cross product is vectors multiplication method, which is contrasted to the dot product because of result in another vector instead of scalar value. What is more, the new vector is perpendicular to both vectors, which are provided in calculations [6].

The notation for the cross product is:

$$\mathbf{a} \times \mathbf{b}$$

However, it is vast to know that the cross product is not communicative:

$$\mathbf{a} \times \mathbf{b} \neq \mathbf{b} \times \mathbf{a}$$

$$\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$$

Further, the mathematical concept of a determinant is useful in the case of the cross product.

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} c_x & c_y & c_z \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

Then, to find the components of the cross product - vector $\mathbf{c}$ - for vectors $\mathbf{a}$ and $\mathbf{b}$ the following formula is required:

$$\mathbf{c} = \mathbf{a} \times \mathbf{b} = (a_y b_z - a_z b_y, a_x b_z - a_z b_x, a_x b_y - a_y b_x)$$

Where each component of the vector $\mathbf{c}$:

$$c_x = a_y b_z - a_z b_y$$

$$c_y = a_x b_z - a_z b_x$$

$$c_z = a_x b_y - a_y b_x$$

However, to check if the calculation is correct and the normal vector is perpendicular it is required to define a new vector completely laying on the surface by adding vectors pointing to the plane and the using dot product formula [9]. The new vector is solved by the formula for adding vectors:

$$\mathbf{p}_2 - \mathbf{p}_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

A dot product is a scalar value, which is gained by a projection of one vector onto another. The basic idea is how much one vector is pointed in the same direction as the other vector. Analysed vectors can be pointed in a positive or negative direction. Otherwise, the vectors are orthogonal if the dot product is equal to 0 [9]. The formula for calculating dot product is:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

Dot Product: Piece by Piece

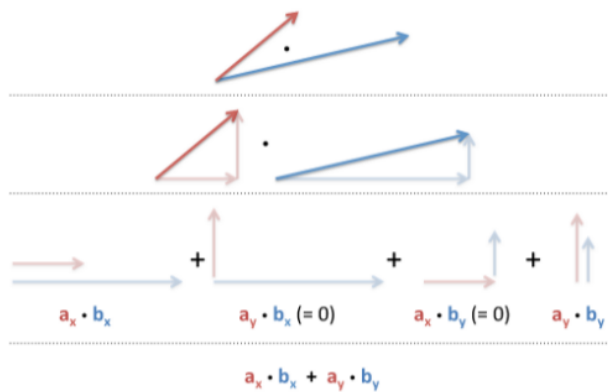


Fig.20 The visualisation of a dot product step by step (<https://betterexplained.com/articles/vector-calculus-understanding-the-dot-product/>).

Accordingly, the dot product of $(p_2 - p_1)$ and the normal vector must be equal to 0 [9]. Thus, the vector equation of the plane is:

$$\mathbf{n} \cdot (\mathbf{v}_2 - \mathbf{v}_1) = 0$$

Hence, it is easier to check a normal vector for the plane by applying the scalar equation of the plane formula, which is:

$$a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1) = 0$$

Where vector $\mathbf{n} = (a, b, c)$.

```

1  #pragma once
2  #include "cVector.h"
3
4
5  class cPlane : public cVector
6  {
7  public:
8      // constructor
9      cPlane();
10
11      cVector getPlaneNormal(cVector &, cVector &);
12      void checkIfPointIsContainedInPlane(cVector &crossProduct, cVector &p);
13      cVector pointP0;
14
15      ~cPlane();
16
17 };

```

Fig.21 The header file of the plane class. The cPlan class is inherited from cVector class as many operations for vectors are included for this task.

```

1  #include "cPlane.h"
2  #include <iostream>
3
4  using namespace std;
5
6  // Set the default value for the point on the surface
7  cPlane::cPlane()
8  {
9      // Set the known vector pointing to the surface.
10     // Coordinates for P0
11     pointP0 = cVector(1, -2, 0);
12 }
13
14 // Find the plane normal
15 cVector cPlane::getPlaneNormal(cVector &v1, cVector &v2)
16 {
17     return getCrossProduct(v1, v2);
18 }
19
20 // Check if a point is lying on the plane
21 void cPlane::checkIfPointIsContainedInPlane(cVector &crossProduct, cVector &p)
22 {
23     // Calculate the dot product
24     float dotProduct = getDotProduct(crossProduct, pointP0, p);
25     cout << "Dot Product:" << dotProduct << endl;
26
27     // If dot product is not 0 then a point is not perpendicular to the normal
28     if (dotProduct != 0)
29     {
30         cout << "The given point is out of plane!!!" << endl;
31         cout << "The normal vector perpendicular to the surface is:" << endl;
32         cout << "N:" << crossProduct.getValueXYZByIndex(0) << " " << crossProduct.getValueXYZByIndex(1) << " " << crossProduct.getValueXYZByIndex(2) << " " << endl;
33     }
34     else
35     {
36         cout << "The given point is contained in the plane." << endl;
37         cout << "The normal vector perpendicular to the surface is:" << endl;
38         cout << "N:" << crossProduct.getValueXYZByIndex(0) << " " << crossProduct.getValueXYZByIndex(1) << " " << crossProduct.getValueXYZByIndex(2) << " " << endl;
39     }
40 }
41
42
43 cPlane::~cPlane()
44 {
45 }
46

```

Fig.22 The structure of the cPlane class, which is responsible for the calculation of the plane normal, setting the point P0 and checking if the provided point is perpendicular to the surface.

```

100 void getNormalVectorOfPlane()
101 {
102     // Create variables for storing points
103     cVector *point0, *point1, *point2 = NULL;
104     point0 = new cVector();
105
106     // Initialize the plane
107     cPlane *plane = NULL;
108     plane = new cPlane;
109
110     // Save the P0 point
111     *point0 = plane->pointP0;
112
113     cout << "Value0 is the point set on the plane?" << endl;
114     cout << "P0 = " << fixed << setprecision(3) << " [" << point0->getValueXYZByIndex(0) << " [" << point0->getValueXYZByIndex(1) << " [" << point0->getValueXYZByIndex(2) << "]" << endl;
115
116     // Set two other points which are part of the plane
117     float x1, y1, z1;
118     cout << "Provide coordinates for the second point set on the plane?" << endl;
119     x1 = readNumber("x1: ");
120     y1 = readNumber("y1: ");
121     z1 = readNumber("z1: ");
122
123     point1 = new cVector(x1, y1, z1);
124
125     float x2, y2, z2;
126     cout << "Provide coordinates for the third point set on the plane?" << endl;
127     x2 = readNumber("x2: ");
128     y2 = readNumber("y2: ");
129     z2 = readNumber("z2: ");
130
131     point2 = new cVector(x2, y2, z2);
132
133     // Display coordinates of the other points
134     cout << "The coordinates of the second point are?" << endl;
135     cout << "P1 = " << fixed << setprecision(3) << " [" << point1->getValueXYZByIndex(0) << " [" << point1->getValueXYZByIndex(1) << " [" << point1->getValueXYZByIndex(2) << "]" << endl;
136
137     cout << "The coordinates of the second point are?" << endl;
138     cout << "P2 = " << fixed << setprecision(3) << " [" << point2->getValueXYZByIndex(0) << " [" << point2->getValueXYZByIndex(1) << " [" << point2->getValueXYZByIndex(2) << "]" << endl;
139
140     // Form vectors parallel to the surface
141     cVector *vector1, *vector2 = NULL;
142     vector1 = new cVector;
143     *vector1 = vector1->formVector(*point0, *point1);
144     cout << "The first formed vector?" << endl;
145     cout << "VP1 = " << fixed << setprecision(3) << " [" << vector1->getValueXYZByIndex(0) << " [" << vector1->getValueXYZByIndex(1) << " [" << vector1->getValueXYZByIndex(2) << "]" << endl;
146
147     vector2 = new cVector;
148     *vector2 = vector2->formVector(*point0, *point2);
149     cout << "The second formed vector?" << endl;
150     cout << "VP2 = " << fixed << setprecision(3) << " [" << vector2->getValueXYZByIndex(0) << " [" << vector2->getValueXYZByIndex(1) << " [" << vector2->getValueXYZByIndex(2) << "]" << endl;
151
152     cVector *point3 = NULL;
153
154     // Provide a point to check if it is a part of the plane
155     float x3, y3, z3;
156     cout << "Provide coordinates to check if a point is perpendicular?" << endl;
157     x3 = readNumber("x3: ");
158     y3 = readNumber("y3: ");
159     z3 = readNumber("z3: ");
160
161     point3 = new cVector(x3, y3, z3);
162
163     // Set a normal for the plane
164     cVector planeNormal = plane->getPlaneNormal(*vector1, *vector2);
165
166     // Check if the given point is set on the plane and display the plane normal
167     plane->checkIfPointIsContainedInPlane(planeNormal, *point3);
168
169     delete vector1;
170     delete vector2;
171     delete point0;
172     delete point1;
173     delete point2;
174     delete point3;
175     delete plane;
176 }
177

```

Fig.23 The functionality of the second part of the task is implemented into the main function. Thereby, a user is able to provide input to the program and analyse the results.

Attempt	Test Values	Description	Expected Result	Actual Result	Pass/ Fail
1	a = (-2, 0, 1) b = (-2, 1, -1)	Calculating the plane normal as a product of two vectors.	x = -3 y = 4 z = 2	x = -3 y = 4 z = 2	Pass
2	a = (3, -2, -1.2) b = (33, 5, -0.1)	Calculating the plane normal as a product of two vectors.	x = 32 y = 7 z = -0.1	x = 32 y = 7 z = -0.1	Pass

```

The coordinates of the second point are:
P1 = (3.000,-2.000,-1.200)
The coordinates of the second point are:
P2 = (33.000,5.000,-0.100)
The first formed vector:
P0P1 = (2.000,0.000,-1.200)
The second formed vector:
P0P2 = (32.000,7.000,-0.100)
Provide coordinates to check if a point is perpendicular:
x: 11
y: 22
z: 6
Dot Product: 1154.800
The given point is out of plane!!!
The normal vector perpendicular to the surface is:
N: (8.400,38.200,14.000)

```

```

The first formed vector:
P0P1 = (-2.000,2.000,1.000)
The second formed vector:
P0P2 = (-2.000,1.000,-1.000)
Provide coordinates to check if a point is perpendicular:
x: 3
y: 2
z: 5
Dot Product: 16.000
The given point is out of plane!!!
The normal vector perpendicular to the surface is:
N: (-3.000,4.000,2.000)

```

Fig.24 The tests and their results.

Attempt	Test Values	Description	Expected Result	Actual Result	Pass/Fail
1	a = (2, 4, 1) b = (4, 8, 4)	Calculating the dot product of two vectors.	-9	-9	Pass
2	a = (3, -2, -1.2) b = (33, 5, -0.1)	Calculating the dot product of two vectors.	1154.8	1154.8	Pass

```

The coordinates of the second point are:
P1 = (3.000,-2.000,-1.200)
The coordinates of the second point are:
P2 = (33.000,5.000,-0.100)
The first formed vector:
P0P1 = (2.000,0.000,-1.200)
The second formed vector:
P0P2 = (32.000,7.000,-0.100)
Provide coordinates to check if a point is perpendicular:
x: 11
y: 22
z: 6
Dot Product: 1154.800
The given point is out of plane!!!
The normal vector perpendicular to the surface is:
N: (8.400,38.200,14.000)

```

```

The first formed vector:
P0P1 = (-2.000,2.000,1.000)
The second formed vector:
P0P2 = (-2.000,1.000,-1.000)
Provide coordinates to check if a point is perpendicular:
x: 3
y: 2
z: 5
Dot Product: 16.000
The given point is out of plane!!!
The normal vector perpendicular to the surface is:
N: (-3.000,4.000,2.000)

```

Fig.25 The tests and their results.

Task 3

The last task of the assignment is to perform a sphere to sphere collision simulation in a computer program. It is required to implement a code to detect collision event and analyse the results. Moreover, the vector class can be used as a base for further calculations.

A collision is when one objects are intersected with the other one – the distance between them is below the point of tolerance. Collision detection is a way of applying computational geometry techniques to detect collisions and collision response is a simulation and effect of that event. In games, collisions are a fundamental of physics in game engines [10] [11] [12].

There are many methods of checking the intersection between objects, due to a variety of needs, such as used shapes or accuracy of detection. A performance of calculation is depended from the precision of operation. However, the analysed method, called a bounding sphere collision detection technique, is one from the simplest to implement and its calculation is not

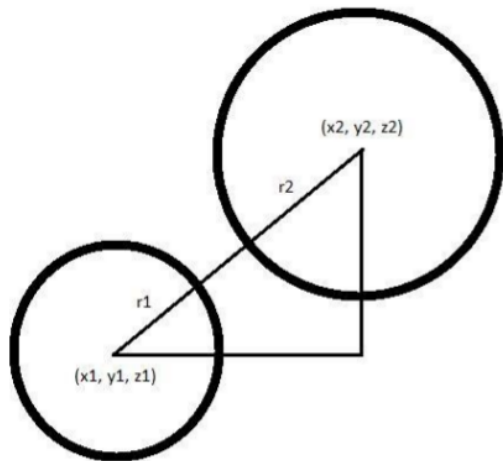
overloaded the system performance. Accordingly, a sphere collider can be quite inaccurate if they are used to non-regular shapes [10] [11] [12].

The special sphere class has been created to solve the task. It is a description of the attributes of a sphere. The class is inherited from vector class as calculation are based on vectors. Nevertheless, a mass, velocity and acceleration are not included for this task.

Furthermore, the collision detection algorithm is made of finding a distance between two points method allocated in vector class. The central point is declared by the user for each created sphere as well as the radius which is pointed from the centre of the object to the surface. Due to this information, the simulation can be performed. If the sum of radiuses is equal or higher than the result of distance calculation, then spheres are intersected. In other cases, there is no collision.

The distance between two central points of spheres is calculated by the equation:

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



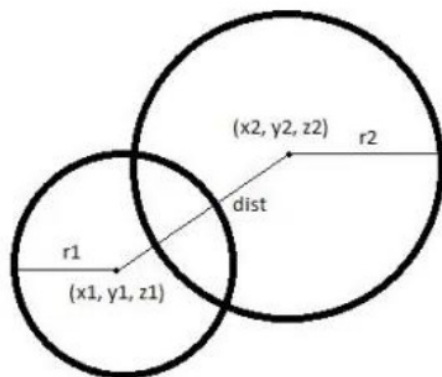


Fig.26 The figures are demonstrating visualisation of the used algorithm concept (<https://vpythontutorials.wordpress.com/2014/05/08/collision-detection-between-two-rectangular-boxes/>).

```

1  #pragma once
2  #include "cVector.h"
3
4  class cSphere : public cVector
5  {
6  public:
7      cSphere();
8      ~cSphere();
9      cVector centralPoint;
10     void setSphere(float x, float y, float z, float r);
11     float getRadius();
12
13 private:
14
15     float radius;
16
17 };

```

```

1  #include "cSphere.h"
2
3
4  cSphere::cSphere(void)
5  {
6      // Set default values for a sphere
7      centralPoint = cVector();
8      radius = 0;
9  }
10
11 // Set a sphere
12 void cSphere::setSphere(float x, float y, float z, float r)
13 {
14     centralPoint = cVector(x, y, z);
15     radius = r;
16 }
17
18 // Get radius of the sphere
19 float cSphere::getRadius()
20 {
21     return radius;
22 }
23
24
25 cSphere::~~cSphere(void)
26 {
27
28 }

```

Fig.27 The centre point and radius of a sphere can be defined by each object of the sphere class.

```

99 void performTask3()
100 {
101     cout << "Initialise two spheres and check if they are colliding each other." << endl;
102     cSphere *sphere1 = NULL;
103     sphere1 = new cSphere();
104     setSphereValues(sphere1);
105
106     cSphere *sphere2 = NULL;
107     sphere2 = new cSphere();
108     setSphereValues(sphere2);
109
110     // Get distance between central points of the spheres
111     float distance = sphere2->centralPoint.getDistanceBetweenTwoPoints(*sphere1);
112     // Get sum of radiuses of the spheres
113     float sumRadiuses = sphere1->getRadius() + sphere2->getRadius();
114
115     cout << "The distance is: " << distance << endl;
116     cout << "The sum of radiuses is: " << sumRadiuses << endl;
117
118     // Check if collision is true
119     if (sumRadiuses >= distance)
120     {
121         cout << "Collision occurred!" << endl;
122     }
123     else
124     {
125         cout << "Non collision found!" << endl;
126     }
127
128     delete sphere1;
129     delete sphere2;
130 }

```

Fig.28 Checking the collisions is simulated in the main class by choosing an appropriate task.

Attempt	Test Values	Description	Expected Result	Actual Result	Pass/Fail
1	P1 = (0, 0, 0) r1 = 5 P2 = (0, 6, 0) r2 = 1	Checking if the spheres have collided.	Collision is detected.	Collision is detected.	Pass
2	P1 = (-2, 4, -0.5) r1 = 2 P2 = (-2, -4, 1) r2 = 4	Checking if the spheres have collided.	Collision is detected.	Collision is detected.	Pass
3	P1 = (0, 0, 0) r1 = 5 P2 = (0, 6, 6) r2 = 1	Checking if the spheres have collided.	Collision is not detected.	Collision is not detected.	Pass
4	P1 = (0, 0, 0) r1 = 0	Radius cannot be zero or below.	Ask user to put radius again.	Ask user to put radius again.	Pass

```
Initialise two spheres and check if they are colliding each other.  
Provide coordinates for the center of the sphere:  
x: 0  
y: 0  
z: 0  
Set the radius for the sphere:  
radius = 5  
Provide coordinates for the center of the sphere:  
x: 0  
y: 6  
z: 0  
Set the radius for the sphere:  
radius = 1  
The distance is: 6  
The sum of radiuses is: 6  
Collision occurred!
```

```
Initialise two spheres and check if they are colliding each other.  
Provide coordinates for the center of the sphere:  
x: -2  
y: 4  
z: -0.5  
Set the radius for the sphere:  
radius = 1  
Provide coordinates for the center of the sphere:  
x: -2  
y: -4  
z: 1  
Set the radius for the sphere:  
radius = 4  
The distance is: 4.58258  
The sum of radiuses is: 5  
Collision occurred!
```

```
Initialise two spheres and check if they are colliding each other.  
Provide coordinates for the center of the sphere:  
x: 0  
y: 0  
z: 0  
Set the radius for the sphere:  
radius = 5  
Provide coordinates for the center of the sphere:  
x: 0  
y: 6  
z: 6  
Set the radius for the sphere:  
radius = 1  
The distance is: 8.48528  
The sum of radiuses is: 6  
Non collision found!
```

```
Initialise two spheres and check if they are colliding each other.  
Provide coordinates for the center of the sphere:  
x: 0  
y: 0  
z: 0  
Set the radius for the sphere:  
radius = 0  
radius = -1  
radius = -10  
radius = 0  
radius = 5
```

Fig.29 The tests and their results.

Summary

In conclusion, the assignment was concentrated on getting knowledge about essential mathematics and physic and using it in practice. Therefore, all tasks have been analysed and attempted to accomplish. What is more, practising programming skills in connection with studying theory of physics in games is resulted in a better understanding of the topic. Describing physical events in computer programming mathematically is a powerful tool, which is used by games developers to create better and more realistic physics simulations and scientists to provide more accurate results or solve complicated problems.

References

- [1] P. Dawkins, "Paul's Online Notes," 12 March 2018. [Online]. Available: <http://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx>.
- [2] San Joaquin Delta College, "Numerical Methods for Solving Differential Equations," 2017. [Online]. Available: <http://calculuslab.deltacollege.edu/ODE/7-C-1/7-C-1-h-c.html>. [Accessed 2 February 2020].
- [3] "Numerical Integration in Games Development," [Online]. Available: <https://jdickinsongames.wordpress.com/2015/01/22/numerical-integration-in-games-development-2/#comments>. [Accessed 2 February 2020].
- [4] H. Serrano, "Harold Serrano - Game engine development," 15 December 2016. [Online]. Available: <https://www.haroldserrano.com/blog/algorithms-in-game-engine-development>. [Accessed 3 February 2020].
- [5] G. Fiedler, "Integration Basics," 1 June 2004. [Online]. Available: https://gafferongames.com/post/integration_basics/. [Accessed 3 January 2020].
- [6] D. J. B. K. A. Stroud, Foundation Mathematics, London: Palgrave MacMillan, 2009, pp. 377-390.
- [7] D. F. a. D. Q. Nykamp, "An introduction to vectors," [Online]. Available: https://mathinsight.org/vector_introduction. [Accessed 4 February 2020].
- [8] P. Dawkins, "Section 6-3 : Equations Of Planes," 5 March 2018. [Online]. Available: <http://tutorial.math.lamar.edu/Classes/CalcII/EqnsOfPlanes.aspx>. [Accessed 4 February 2020].
- [9] The University of Sydney, "13. Planes in space," 9 November 2009. [Online]. Available: <http://www.maths.usyd.edu.au/u/MOW/vectors/vectors-13/v-13-1.html>. [Accessed 4 February 2020].
- [10] M. J. J. van den Heuvel, "Pool Hall Lessons: Fast, Accurate Collision Detection Between Circles or Spheres," [Online]. Available: https://www.gamasutra.com/view/feature/131424/pool_hall_lessons_fast_accurate_php?print=1. [Accessed 4 February 2020].
- [11] N. Souto, "Video Game Physics Tutorial," [Online]. Available: <https://www.toptal.com/game/video-game-physics-part-i-an-introduction-to-rigid-body-dynamics>. [Accessed 5 February 2020].
- [12] O. Dopertchouk, "Simple Bounding-Sphere Collision Detection," 12 November 2000. [Online]. Available: <https://www.gamedev.net/tutorials/programming/math-and-physics/simple-bounding-sphere-collision-detection-r1234/>. [Accessed 5 February 2020].

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

70/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

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

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

GRADING FORM: SMU GENERIC_GJ

**BARTOSZ
GALANTOWICZ**

69.50

RESEARCH-INFORMED LITERATURE



Refs included - many are online - try to avoid on-line refs and stick to established texts, papers, etc..

6

Essential topics covered and discussed. Good.

KNOWLEDGE AND UNDERSTANDING

	A good understanding demonstrated. covered all areas required. Code shows good skills in C++ and SE practices.	14
---	--	-----------

ANALYSIS

	Results are graphed. REview of methods and some comparison included. Test schedule provided. OOA and OOD skills in evidence.	13.5
---	--	-------------

PRACTICAL APPLICATION AND DEP.

	Only Euler documented. Other Tasks implemented as required. Good SE practices demonstrated.	30.5
---	---	-------------

SKILLS FOR PROFESSIONAL PRAC.

	Report strucutred. Some sections could be more precise/clear in the use of language, otherwise v good. Avoid on line refs. Place tables of results in appendices.	5.5
---	---	------------